University of Alberta

Token-based Graphical Password Authentication

by

John Charles Gyorffy

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Examining Committee

ABSTRACT

Given that phishing is an ever increasing problem, a better authentication system than the current alphanumeric system is needed. Because of the large number of current authentication systems that use alphanumeric passwords, a new solution should be compatible with these systems. We propose a system that uses a graphical password deployed from a Trojan and virus resistant embedded device as a possible solution. The graphical password would require the user to choose a family photo sized to 441x331 pixels. Using this image, a novel, image hash provides an input into a cryptosystem on the embedded device that subsequently returns an encryption key or text password. The graphical password requires the user to click five to eight points on the image. From these click-points, the embedded device stretches the graphical password input to a 32-character, random, unique alphanumeric password or a 256-bit AES key. Each embedded device and image are unique components in the graphical password system. Additionally, one graphical password can generate many 32-character unique, alphanumeric passwords using its embedded device which eliminates the need for the user to memorize many passwords.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY OF TERMS

USB	Universal Serial Bus
Character	A text value that includes most items on the U.S. keyboard including both upper and lower case.
Payload	The data from the user that is carried with system-level stream of data.
RNG	Random Number Generator- A true random number generating application that produces numbers that are statistically independent from the next.
PRNG	Pseudorandom Number Generator- Number generator that may not produce independent events from the next.
Entropy	The random and statistical independence of bits in a stream of bytes or the.
Password Entropy	The random and statistical independence of a sequence of items that form a password.
Hash value	A smaller representation of a much larger source of data.
Image hash value	A smaller, unique representation of an image.
Cryptographic hash value	A hash value that is statistically impossible to reverse in order to define the source data.
Hash message	The data input into the cryptographic hash function.
Hash message digest	The hash value returned from the cryptographic hash function.
Nonce	Cryptographic nonce**-**Number Used Once- Ensures random encryption output.
Calculated Nonce	A number derived from a shared secret between two parties using a nonce value from the party sending a message. Used to verify the receiving party.
ARP	Address Resolution Protocol
DNS	Domain Name System
API	Application Programming Interface
DLL Injection	A method to run code within the address space of an unsuspecting process by forcing it to load a dynamic-link library.
Brute force	Guesing a value based on iterating each possible permutation in a defined set.
Hot-spot	An area of common focus on a graphical password image that affects entropy
Out-of-band	An alternate secure path to exchange information outside of the main path.
AES	Advanced Encryption Standard

CHAPTER 1 INTRODUCTION

Given the efficacy of cyber attacks that steal, or phish, a user's login credentials, it is imperative to provide a more secure means of safeguarding passwords than what is common today. Additionally, any useful solution must use the existing Web infrastructure, be cost effective, and easy to use. Attacks that steal login information fall into the definition of phishing. As defined by the Anti-Phishing Working Group (APWG), phishing is a criminal means of stealing a victim's digital personal identity such as usernames and passwords that can lead to fraudulent activity [1]. The impetus to steal a user's login credentials is frequently financial fraud. A Gartner study estimated that all varieties of phishing attacks in the U.S. cost $3.2 billion and affected 3.6 million adults in 2007 [2]. According to APWG, between 92 and 94 percent of phishing activity is directed toward the financial industry [1, 3]. It appears the trend is getting worse despite consumer education. Alnajim and Munro found that although users can be better educated in phishing prevention, finding the most effective way to educate users can be problematic [4].

One common way to phish is for an attacker to install a key-logging Trojan. A Trojan can spread itself, like a virus, to other programs or host machines on a network and pose as a legitimate program [5]. Though key-loggers may be detected by some anti-malware software, this can only occur if a file or memory signature of the Trojan already exists in some anti-malware's database. However, key-loggers themselves have no specific signature [6]. According to APWG, there was a record 430 unique key-logging applications detected in malicious code in Q1 of 2008, an increase of 18% recorded during the same period in 2007 [1]. Key-loggers need not be a software application; rather, key-presses can be captured with a small piece of hardware purchased at some retail stores [7].

Phishing usernames and passwords can also occur while using the Internet Public Key Infrastructure (PKI) cryptography, the de facto standard for encrypting sensitive data between clients and Web servers using the “secure” protocol HTTPS. The encryption strength used is unarguably quite hard to attack; however, as with any encryption, there is always a possibility of an attack on the infrastructure. Man-in-the-middle (MITM) attacks can intercept exchanges of usernames, passwords, credit card numbers, and other sensitive data by exploiting the Web browser’s trust of public keys. The attack can occur either by DNS poisoning or ARP spoofing [8-10]. Additionally, a simple MITM attack can involve tricking victims into using a fake, malicious Web site that has a URL which is unnoticeably similar to the authentic one. Even though some Web browsers warn users of a possible MITM attack, Schechter et al. demonstrated that users tend to ignore some very obvious security warnings [11].

To prevent some common phishing attacks, we propose a system that is resistant to Trojans and viruses, produces a stronger password for authentication than an 8-character alphanumeric password, and is easier to use when multiple alphanumeric passwords are needed. Additionally, we propose that this system can use 256-bit, Cipher Block Chaining (CBC), AES cryptography, using random, unique symmetric keys, to prevent MITM attacks. We show a proposed AES key generation scheme and a key exchange protocol. The crux of the proposed system is a graphical password comprised of a 441x331 pixel image the user chooses from family photos. Using the uniqueness of this image, a novel, resulting image hash and the click-points gathered from the user, along with some random numbers from an embedded device, will be the message input into a 256-bit secure hash algorithm (SHA-256). The resulting 256-bit message digest will be used for either a 256-bit AES key or a 32-character alphanumeric password.

The graphical password system proposed is built directly into a special client Web browser embedded in a USB device that uses an inexpensive microcontroller and read-only, protected flash memory. Unlike other embedded systems, we propose a system that will not store any sensitive information in the device; rather, flash memory is used to store the graphical password image the user chose and some truly random numbers generated at a factory. Since no sensitive information is stored in the device, there is no need for expensive hardware to guard against side channel attacks and other hardware cryptanalysis schemes [12-14]. Therefore, the graphical password must be used with the device every time an AES key or text password is needed. The embedded device ensures that the user cannot generate an AES key or text password without it because the device provides the random numbers to the SHA-256 function in the device. This decreases fraud because only the owner of the device can obtain their password. All communication with the device is encrypted by 256-bit, AES, CBC, cryptography using file I/O. Though the client Web application is deployed from a read-only device, any viral attack on the client, as it loading, or a Trojan client application run from outside the device, are prevented by a novel scheme that uses partial executable checksums.

CHAPTER 2 SYSTEM BACKGROUND

2.1 Analysis of Password Stealing URLs

According to APWG, in its Activity Trends Report for Q1 of 2008, password stealing code downloaded from rogue URLs has been increasing consistently over time (Figure 1) [1]. Several monthly figures for 2008 were almost double that of 2007 (Figure 2) [3]. An increase in attacks that trick users with fake Web site brands in order to phish sensitive information, including passwords, has also been on the rise (Table 1). The password stealing URLs are primarily targeting the financial services sector (Table 2). This seems to suggest that attackers are phishing sensitive account information so they can commit fraud.

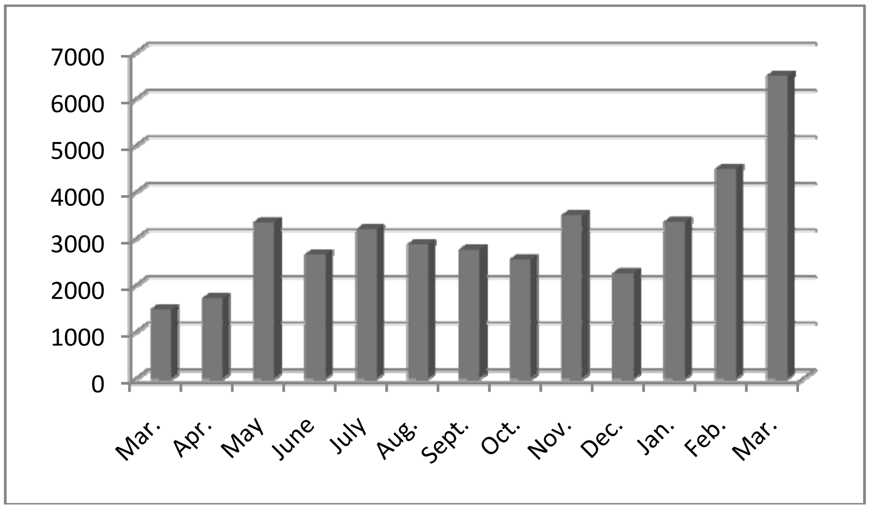


Figure 1: APWG-URLs with password stealing code in Q1 of 2008 [1].

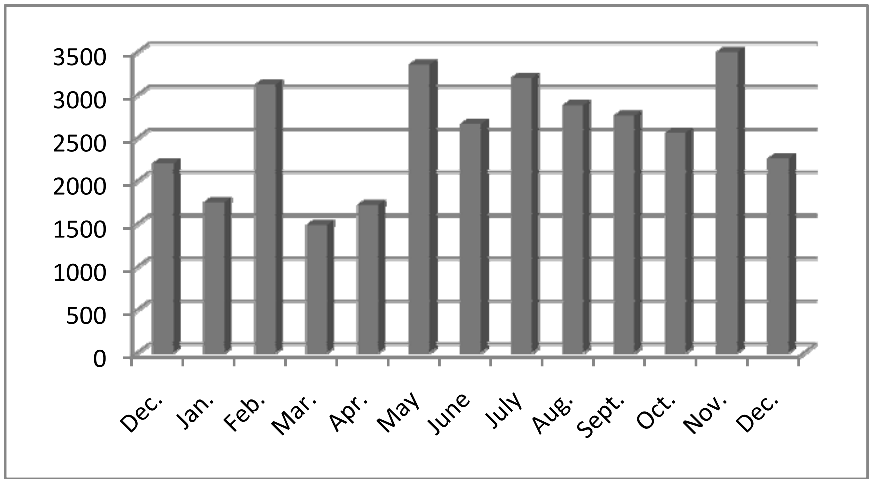


Figure 2: APWG-Password stealing malicious code URLs reported in 2007 [2].

Table 1: APWG- Statistical highlights for Q1 2008 [1].

Months Reported	**January**	**February**	**March**
Number of unique phishing reports received	29,284	30,716	25,630
Number of unique phishing sites received	20,305	36,002	24,908
Number of brands hijacked by phishing campaigns	131	139	141
Country hosting the most phishing websites	US	US	US
Contain some form of target name in URL	28.3%	23.2%	26.1%
No hostname; just IP address	5.5%	13.2%	4%
Percentage of sites not using port 80	.81%	.45%	.49%
Longest time online for website	31 days	29 days	31 days

Table 2: APWG-Industries attacked as a percent of all phishing attacks (2008) [1].

Industries	**January**	**February**	**March**
Financial Services	92.4%	94.2%	92.9%
Retail	1.5%	1.4%	1.4%
ISPs	3.8%	2.2%	1.4%
Government and Others	2.3%	2.2%	4.3%

2.2 Web Site Forgery Countermeasures

A common form of phishing attack, as mentioned by APWG, is when a user is fooled by an authentic looking Web site forgery of a brand they trust. The unsuspecting user will type their user name and password into a malicious (and fake) site revealing the necessary authentication credentials to the attacker for the authentic Web site. One way to mitigate these attacks is to allow the user to setup a site-authentication image. The site-authentication image is an image that users configure with their bank, or other institution, and is their own personal image and shared secret. The user should know that this image indicates that they are inputting their password into an authentic Web site. If the user doesn't see this image, they should not enter their password. Bank of America uses a patented site-authentication image scheme called *SiteKey* that will accept a user's moniker, display a security question known only to the user, and if correct, displays the user's personal image on the password page [16]. The user is instructed by the login Web page to make sure that the image is correct before entering their final password to gain entry.

Another security measure to prevent phishing with Web page forgeries is to warn the user with HTTPS indicators. When using HTTPS, or the PKI secure Web protocol, the URL will indicate the protocol as part of the Web address. Additionally, many browsers will display a little icon to indicate a secure session. Warnings can also be brought to the user's attention that either the public key is not signed by a trusted certificate authority or the URL does not match the URL hashed in the public key. Mozilla, one such Web browser, defaults to warn users that the site certificates are either unsigned, expired, or does not match the intended URL. However, a user can configure the security settings to reduce or eliminate any warnings [17, 18].

Schechter et al. studied the efficacy of some security indicators that warn users of site-forgery phishing attacks [11]. The Schechter study measured the responses of three groups when presented with three increasingly noticeable security indicators while logging into a bank's Web page. Three groups were monitored (Table 3):

- One group was given no security focus and knew they were not using a real bank.
- One group was given a security focus but knew they were not using a real bank.
- One group believed they were logging into their own bank account but did not know that security was the focus of the study.

A proxy Web server was used to intercept users' requests for a bank login page. The proxy inserted its own Web pages when capturing users' actions.

In the study, one test measured the responses of all the groups when presented with no HTTPS indicators. All participants, 100%, submitted their password when no HTTPS indicators were present. A slightly more obvious security indicator was removed and the responses measured (Table 4). Users were presented with a message indicating an upgrade on the bank's Web site and no site-authentication image. Additionally, no HTTPS indicators were present either. Even so, 92% of the users in the group that were using their own bank account entered their password and 100% of the users in the other groups did as well. The last experiment used an even more obvious security threat indicator: a warning page from Internet Explorer 7 that stated that there was a security certificate problem which could result in the theft of their data. This warning page was presented before the password page and the user could exit before entering a password (Table 5). A majority, 71%, of those that were security primed entered their passwords. Of the users that knew they were entering their password into their real personal account, 36% ignored the warning and entered their password anyway.

Table 3: Users roles in the Schecter et al. warning indicators efficacy experiment. [11].

Group	Name	Characteristic
1	Role playing	Played a role, given no indication that security is focus of study
2	Security primed	Played a role, told that their role was concerned about security
3	Personal account	Used their own account, given no indication that security is focus of study

Table 4: Actions when not presented with a site image (Schecter et al.) [11].

Action	Group 1	Group 2	Group 3	Total
Send password	100%	100%	92%	97%
Did not login	0%	0%	8%	3%

Table 5: Actions when presented an obvious warning page (Schecter et al.) [11].

Action	Group 1	Group 2	Group 3	Total
Send password	56%	71%	36%	53%
Did not login	44%	29%	64%	47%

This research shows that users will regularly ignore Web browser security indicators. Additionally, most browsers can be configured to reduce these indicators. Even if users can learn better anti-phishing behavior, as the attacks become more advanced, identifying which security indicators are effective in their education can be elusive. A study by Alnajim and Munro looked at the effectiveness of seven common indicators. Only one indicator, which was to check the URL carefully, received a 3 out of a scale of 4 in effectiveness. No indicator received a score of 4 out of 4 [4].

2.3 Malware Detection

In addition to users being fooled to give away authentication credentials to attackers, Trojans and worms, or malware, can deploy identity-theft software. Of the possible malware investigated by one study, 46% had remote access [19]. This suggests that malware is uploading user details to the attacker. Software methods can be effective in detecting malware. Anti-malware can scan files or memory for known malware signatures; however, malware can change or mutate the signature polymorphically using various encryption algorithms [20]. Integrity checking files using a checksum can be effective but this strategy relies on anti-malware software having a current database of file checksums. Heuristic checking uses a rule-set developed from measuring the normal operation of a system such as operating system calls and interrupts. Any deviation from a rule-set would reveal a potential threat. However, it is hard to fully indentify all the necessary rules leaving a potential hole that can be exploited [21]. How well do these anti-malware applications perform? The Anti-Malware Test Lab measured the effectiveness of 10 heuristic analyzers on the market in 2008 (Figure 3). The best vendor caught 71% of the infected viruses [22]. A comparable study by AV-Comparatives.org showed a break-down of the virus types and the efficacy of similar vendors. Similarly, the best vendor caught 73% of worms and 63% of Trojan infections [23]. Clearly, a near 30% chance still exists for undetected viruses either because they are too new or simply not prolific enough to be known by anti-malware vendors.

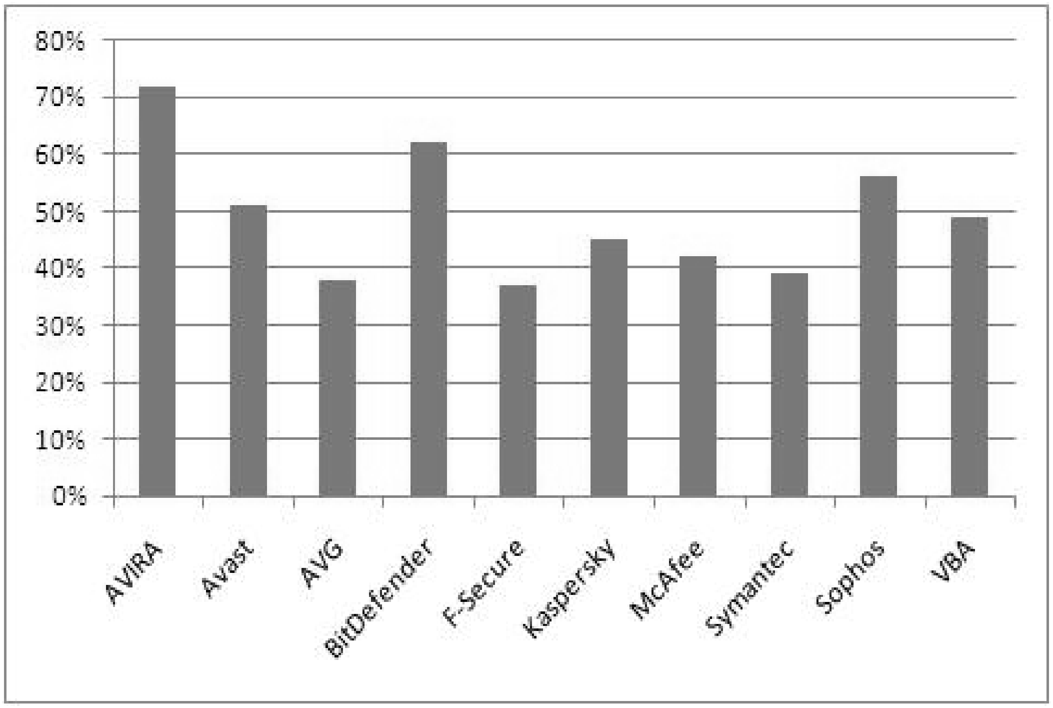


Figure 3: Percent effectiveness of some common heuristic analyzers [21].

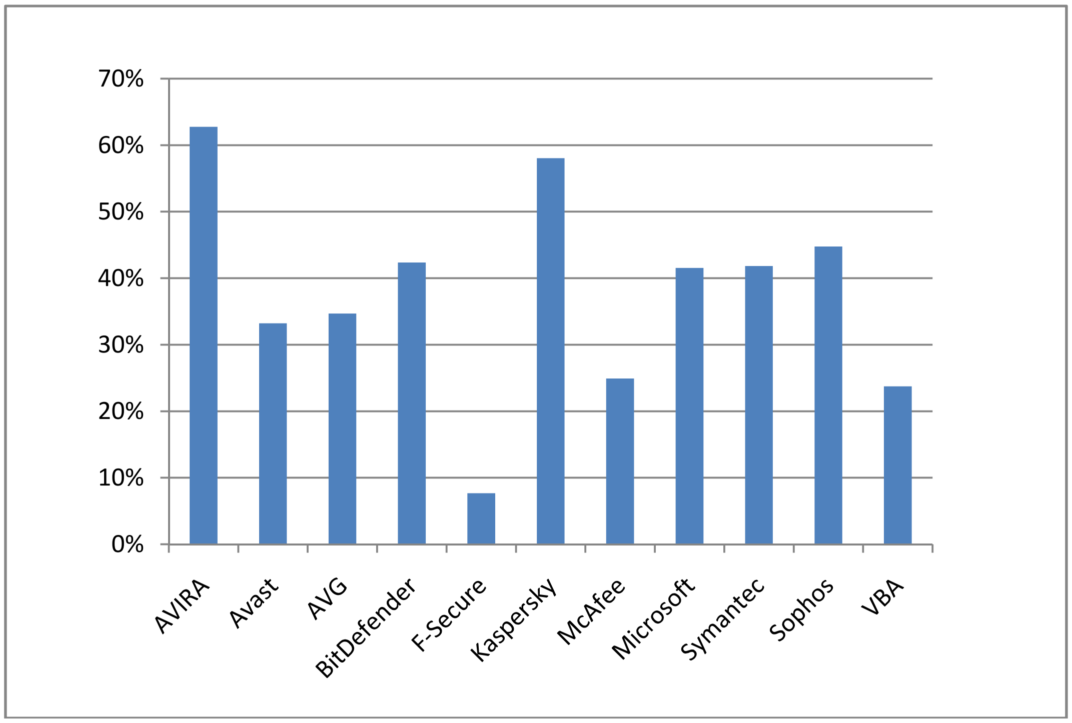


Figure 4: Percent of Trojan infections discovered with some common anti-malware [22].

2.4 Text-Based Passwords: A Low-Entropy Solution in Practice

The theoretical strength of an alphanumeric password can be quantified by its password space: "the total number of distinct passwords that can be created with a given set of characters" [24]. Using a standard U.S. PC keyboard, there are 10 digits, 26 uppercase letters, 26 lowercase letters, and 33 symbols giving a total of 95 possible characters to choose. However, the allowable special characters depend on the authentication system. With an eight-character alphanumeric password, there are 95^8, or 6.6×10^{15}, possible permutations. However, this presupposes an unrelated statistical event in the selection of each character. How random or unrelated these characters are as an ordered combination is a measure of password entropy. Because of the limitations of human memory, users tend to pick easy to remember text-based passwords; consequently, this reduces the entropy [15, 25].

Attackers can use various ways to crack, or guess, a low-entropy password. A dictionary style attack uses a database of commonly used words, phrases, or character sequences to guess the password [26]. For example, dictionary attacks are effective guessing passwords that include people's names, birthdates, common calendar dates, phone numbers, and adjacent keys. Likewise a brute force attack can leverage the weakness of the password length. A study by Yan found that 86% of passwords are case-insensitive and without special characters, leaving only 36 characters in practice. Using the theoretical password space of 36^7, for a seven-character password, Yan used character patterns as a weakness of low entropy passwords. With a program called *Crack*, they computed the time it took to crack various character patterns. With P representing password space and P (6a+1n) meaning six letters and one number, the following table was tabulated (Table 6) [25]. Yan's study found that dictionary attacks were successful on password patterns using three or more adjacent repeating letters or more than five numbers without knowing the semantics of the password. It is clear that users can choose low-entropy passwords but can users be coached or guided to enter high-entropy, hard-to-guess, alphanumeric passwords?

Table 6: Different distribution areas for 7-character passwords (Yan et al.) [25]

Distribution	**Cracking Time (hours)**
P(all random)	102.31
P(7a)	10.49
P(6a+1n)	28.23
P(5a+2n)	32.57
P(4a+3n)	20.88
P(3a+4n)	8.03
P(2a+5n)	1.85
P(1a+6n)	0.24
P(7n)	0.01

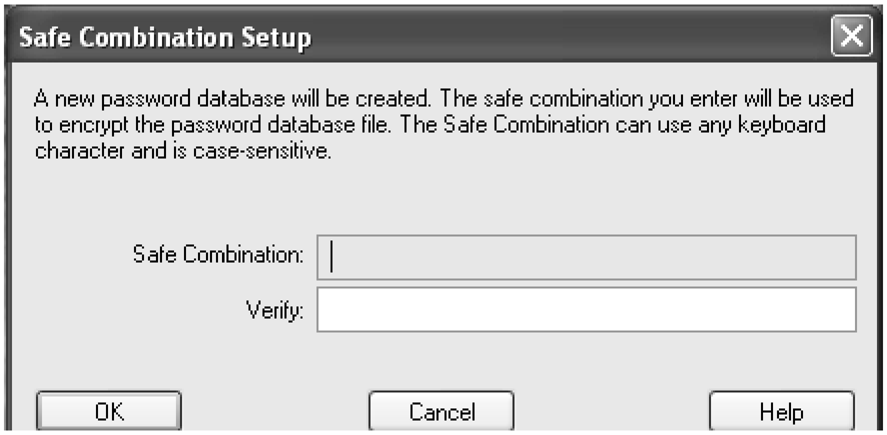
Another study by Yan et al. found that mnemonic-based text passwords were a significant improvement over easy-to-guess passwords. Mnemonic-based passwords use a memorable phrase to select one character to represent a word. The idea is that the resultant password would be random but memorable. Though given instructions on how to use mnemonic passwords, users still had a ten percent non-compliance rate [26]. Kuo et al. conducted a study of the strength of mnemonic-based passwords compared to control passwords that were restricted to eight characters and some special characters [27]. A mnemonic-password dictionary was created from phrases obtained from movies, music and famous quotes off the Internet. By using a dictionary password cracker called *John the Ripper's English dictionary*, they vetted out weak passwords from the group of passwords that did not use mnemonics. Passwords generated from mnemonics were vetted with their mnemonic dictionary. By comparing the strong passwords in the control that were non-mnemonic with the mnemonically derived strong passwords, there was no statistical significance in the graded strength. This suggested that other password persuasive techniques can be just as strong as mnemonics. However, their study also suggested that even mnemonic-passwords are susceptible to mnemonic-dictionary attacks [27]. It seems that it is possible for users to be persuaded into using higher entropy passwords but there is a constant struggle between unmemorable, high-entropy passwords and low-entropy but memorable passwords [28]. Not only is it a struggle to find one good password that is resistant to dictionary attacks, it is also a burden on users to memorize more than one strong password. With an array of login screens on the Web, many users may prefer to memorize one easy password for all sites.

2.5 Password Vaults: A False Sense of Security

Given that a high-entropy password is difficult to remember and that multiple passwords may need to be remembered, users tend to write them down [4]. However, what if these passwords could be stored in encrypted form on some storage media? A possible solution is a secure password vault. Using a vault managing application, these passwords can be stored with the Web address of the corresponding login page. When the User logs into the vault manager, Web addresses can be associated with the encrypted password [29-31]. This has three advantages:

- The user only needs to remember one password to access the password vault.
- The vault manager retrieves the password so the user does not need to remember it.
- The effectiveness of some phishing attacks is reduced.

The idea of a password vault that can assist in storing very long and randomized passwords does help manage stronger passwords. However, there is little reason to believe that this will prevent a phishing attack from retrieving these protected passwords. For instance, to gain entry into the vault, one password is used (Figure 5) [31]. During the setup of the password vault, a key-logger could be actively recording the key strokes. Since Trojans can mimic valid programs, there is also the possibility that a Trojan could mimic the vault manager, record the key strokes, decrypt the passwords, and send them back to the attacker.


Safe Combination Setup

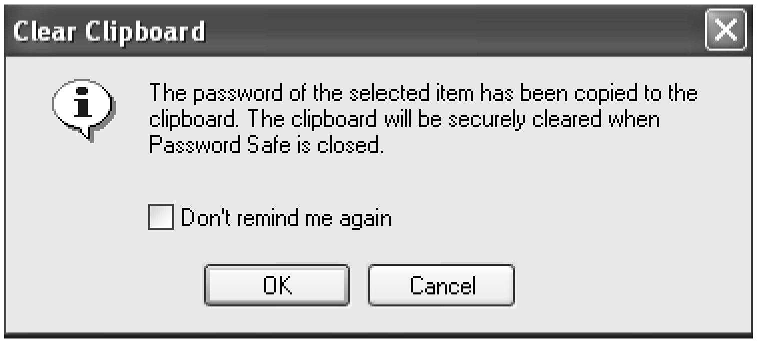
A new password database will be created. The safe combination you enter will be used to encrypt the password database file. The Safe Combination can use any keyboard character and is case-sensitive.

Safe Combination:

Verify:

Figure 5: Password Safe-Master password setup showing text input (open to key-loggers) [31].

Another problem with password vaults is how they inject the text password into the Web browser during login. It may be true that using the vault manager would negate the need to press the keyboard; hence, key-loggers would be ineffective. However, some solutions use the clipboard as a way to transfer the decrypted password to the password field of a login page (Figure 6) [31]. The clipboard is an area that any application on a user’s system can access. This communal area can also be accessed by spyware.


Clear Clipboard

The password of the selected item has been copied to the clipboard. The clipboard will be securely cleared when Password Safe is closed.

☐ Don't remind me again

Figure 6: Password Safe-Copying the password to the clipboard (open to Trojans) [31].

Even how the passwords are encrypted can be of concern. For example, one vendor uses 160-bit encryption instead of a stronger 256-bit AES encryption [30]. Likewise, the encryption key generation may have a weakness allowing for an attacker to extract the master key [32]. By having all the user's passwords concentrated into one area, these passwords become a high-value target for an attacker. As such, any weakness in the encryption key, key storage or the application itself is a plausible target. Storing passwords into one location seems to be quite risky despite the gain in security by using more robust passwords.

2.6 Graphical Passwords: The Picture Superiority Effect

Given these weaknesses, an important question to consider is: are there other forms of input that can be unique enough for authentication? Particularly, is there another mechanism to capture input with current PC-based technology? Users will have a mouse, keyboard, and on some systems a touch screen, as input devices. One possibility is to leverage the unique and highly advanced system of visual recognition by humans. Nelson found that people were better at discriminating between images than words. As such, Nelson was credited with the concept of the Picture Superiority Effect [33]. Humans possess a strong ability to discriminate between subtle differences in images; hence, this could be reason to believe a graphical "password" scheme could be effective. Contrasted with a text-based password scheme, the password space of an image is limited by how many images exist in a collection of images or how many areas one image can be subdivided [36-40].

Image processing relies on the visual working memory (VWM) to retain object information so it can be processed and memorized after the information is removed. Images can have many shapes and formations that may be less familiar than the symbols found on a PC keyboard. Would unfamiliar objects prove to be more difficult to memorize than the alphabet? Research by Chen et al. showed that unfamiliar shapes had equivalent retention in the VWM as well known shapes [34]. Miller studied the limit of human memory with visual object recall [35]. Miller reported that short-term memory had a limitation of around seven objects that could be received, processed or remembered. He noted that if information is organized into a sequences or chunks, this limited could be stretched.

A study by Dhamija and Perrig used a graphical password scheme they named Déjà vu [36]. Déjà vu displayed 20 random images, from a database of many images, with 5 images required to compose the password. With trials that compared 4-digit PINs and Déjà vu passwords, their results showed a better recall of 5 sequential images than with 4-digit PINs (Table 7). A commercial graphical password application called Passfaces™, developed by Real User Corporation, uses a large selection of images depicting human faces of various genders, races, ages, and expressions. Passfaces™ requires a user to learn four faces displayed one at a time with eight random distracter faces [37]. Brostoff and Sasse conducted a study comparing Passfaces™ with text-based passwords. Though their study was not very clear on the strength of the text-based passwords used, there was a significant difference in login attempts. It took nearly a third fewer attempts with Passfaces™ (Table 8) [38].

Table 7: Déjà vu –Percent failed logins (# failed logins/20 participants) [36].

Failed Logins	**4-digit PIN**	**6-character Password**	**Abstract Art**	**Meaningful Photo**
Login (immediate)	5%	5%	0%	0%
Login (after 1 week)	35%	30%	10%	5%

Table 8: Passfaces™ - Statistics for the number of login attempts [38].

Group	**Mean**	**N**	**95% CI**	**Std Err**	**Std Dev**	**Min**	**Max**
Text passwords	33.91	34	27.00/40.83	3.40	19.81	0.00	92.00
Passfaces	12.32	34	9.92/14.73	1.18	6.88	2.00	29.00

Another study, over a longer period of time, by De Angeli et al. suggested a useful classification of graphical passwords into three areas: cognometrics, locimetrics, and drawametrics [39]. Cognometrics is an image recognition scheme, like Passfaces™ or Déjà vu, that challenges a user's recall of a series of discrete images within a series of distracter images. Locimetrics is a graphical password that challenges a user's memory of specific areas or loci on an image. Both cognometrics and locimetrics work with PC based systems; however, drawametrics are graphical passwords that measure the uniqueness of touch screen based drawing activity with a stylus. De Angeli et al. compared two cognometric schemes that used four discrete images with 4-digit PINs in 2,196 authentication attempts over several weeks [39]. In their study, they rejected the picture superiority effect hypothesis but confirmed that spatial coding is a strong factor in the memory recall of visual objects (Figure 7). While the De Angeli study illustrated the fact that cognometic passwords might not be an advantage compared to short numeric sequences, the study did indicate one advantage of graphical passwords that does not exist in text-based passwords: spatial coding [39]. Spatial coding would be very important in locimetric schemes where the user must select unique points on an image as opposed to a sequence of images.

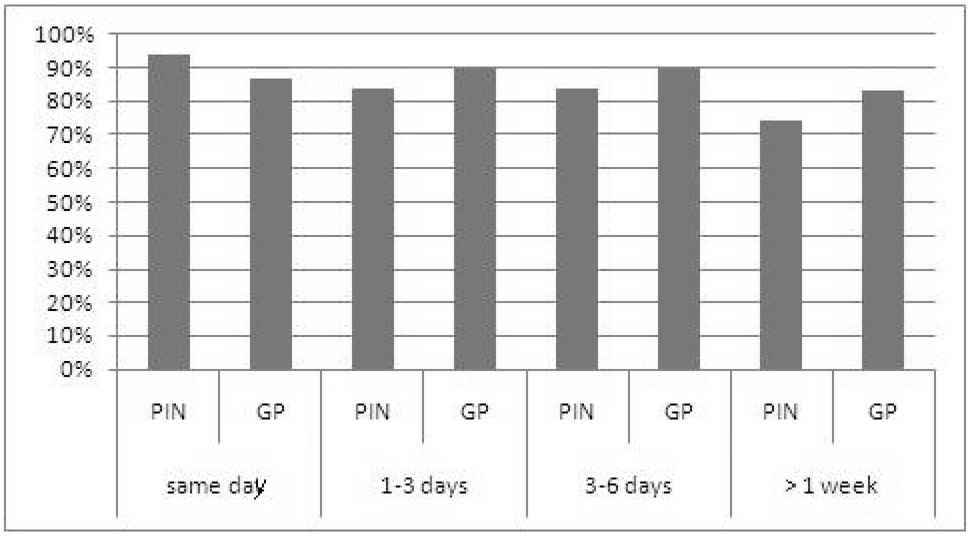


Figure 7: Percentage of correct authentications as a function of system and time (De Angeli et al.) (GP=graphical password; PIN= 4-digit password) [39].

Locimetric schemes, using an image with clickable areas, require a user to differentiate and memorize particular areas of an image. One such design, PassPoints, was evaluated by Wiedenbeck et al. against alphanumeric passwords over a six-week period [40]. In this study, several questions were proposed:

- Are graphics comparable or better than alphanumerics in the creation, learning, performance and retention of passwords?
- Can users feel as secure with graphical as with alphanumeric passwords?

Wiedenbeck et al. noted that unlike alphanumeric password recall, locimetric passwords rely on cued recall. That is, a user might perform better recalling their password by virtue of remembering adjacent image loci. Other than the Web page address, a user is confronted with a blank input box and no clues as to what the password might entail when using an alphanumeric password scheme. However, with locimetric passwords, the user is presented with a familiar image before clicking a series of discrete regions. The user could remember the correct series of mouse clicks on the image by remembering areas of significance on the image. Therefore, it was postulated, that locimetric password recall could be easier than alphanumeric password recall. These results seem to confirm the results of De Angeli et al. [39] that spatial encoding with cued recall is a strength of graphical passwords.

Wiedenbeck et al. [40] compared PassPoints to alphanumeric passwords where the alphanumeric passwords were restricted to the following:

- Eight characters including at least one uppercase letter and one numeric.
- No special characters.
- No previously used passwords or variations of previous passwords.
- The password system enforced length and character encoding conditions.
- The user could view their password as clear text during the learning phase.

It took the alphanumeric group more attempts than the PassPoints group to create a valid password. This may have been because there were no restrictions on where PassPoints users could select areas on the image. Additionally, the alphanumeric group reported having more trouble thinking of a good password. However, there were more incorrect practice submissions with PassPoints (Table 9). When asking users survey questions, a scale of 1 to 7 was used where the lowest number indicated strongly agreeing.

Table 9: Creating alphanumeric passwords vs. PassPoints [40].

Metric	**Mode**	**Mean/ SD**
Total attempts to create	Alphanumeric	1.70/0.18
	PassPoints	1.10/0.07
I did not have much trouble thinking up a password	Alphanumeric	3.30/1.59
	PassPoints	2.35/1.57
It did not take me long to think up a password	Alphanumeric	3.15/1.63
	PassPoints	2.60/1.42
Number of incorrect submissions (learning phase)	Alphanumeric	0.40 /0.68
	PassPoints	4.80 /7.16

To examine whether PassPoints was easier to remember over time compared to alphanumeric passwords, the retention of the two groups were measured in three periods: one week (R1), two weeks (R2), and six weeks (R3). The data recorded over six weeks seemed to show that PassPoints was only slightly statistically easier to recall in R3. However, in the first two weeks, PassPoints did not seem to offer any improvement in recall (Table 10). It seems that PassPoints might be somewhat harder to use than alphanumeric passwords and only slightly more memorable.

Table 10: Means/Std. deviation of number of incorrect password submissions (alphanumeric N =20/graphical N = 20) [40].

Metric	**Mode**	**R1-Mean/ SD**	**R2-Mean/ SD**	**R3-Mean/ SD**
Number of incorrect submissions	Alphanumeric	.25 /0.79	2.20 /2.73	1.75 /2.47
	PassPoints	1.55 /1.57	2.75 /3.88	1.50 /2.80

Chiasson et al. studied PassPoints using seventeen different image types classified by visual clutter, color and content using images with a size of 451x331 pixels and a clickable resolution of 19x19 pixels [41]. They found that different image scenes could show modest differences in login success rates (Table 11). Two particular scenes were investigated: a pool and a car (Figures 8 and 9). The better success rate for the pool scene could be due to the repetitive nature of similar objects in the car scene versus a more natural scene. The study also investigated the effect of changing the clickable size and monitor resolution (Tables 12 and 13). Changing the clickable area tolerance and screen resolution modestly had little statistical difference in login success. Though this study did not compare the effectiveness with alphanumeric passwords, it did elucidate the effect of clickable area tolerance, screen resolution, and image scenes. From the Chiasson et al. study [41], there does seem to be a variation in recall between natural scenes and more cluttered repetitive scenes.

Table 11: Success rate of PassPoints (Chiasson et al.) [41],

Metric	**Pool**	**Car**	**All Image Types**
Login	33/33 (100%)	30/32 (94%)	560/598 (94%)

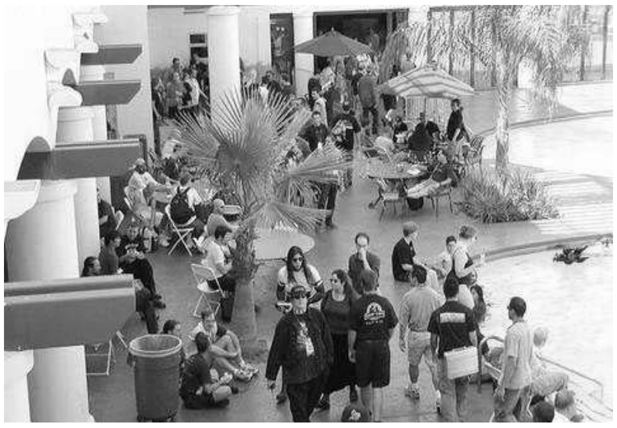


Figure 8: Pool Scene [41].

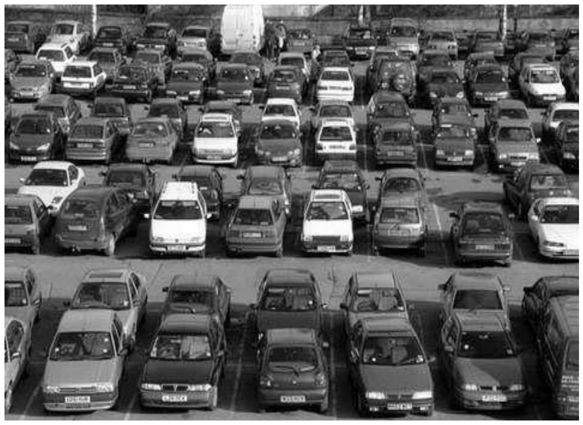


Figure 9: Car Scene [41].

Table 12: Login effect of size of tolerance square on success rate (Chiasson et al.) [41].

Scene	**13x13 Tolerance**	**19x19 Tolerance**
Pool	790/1018 (78%)	671/862 (78%)
Cars	640/790 (81%)	661/773 (85%)

Table 13: Login effect of screen resolution on success rate (Chiasson et al.) [41],

Scene	**Low Res.** **(<= 1 million)**	**High Res.** **(> 1 million)**
Pool	1000/1268 (79%)	460/611 (75%)
Cars	725/875 (83%)	575/687 (84%)

Image sequences found in cognometric schemes seem to offer no improved login success when compared to short, weak text-based passwords or PINs. However, when compared to stronger text-based passwords, the picture superiority effect may enhance recall using cognometric passwords, like Passfaces™, but not to a large degree. However, if a sequence of images is not required when forming a password, the picture superiority affect is much stronger [52]. Locimetric passwords show a slight improvement in login success compared to cognometric and alphanumeric passwords because of cued and spatial recall.

2.7 High Entropy Graphical Passwords

Having a theoretically large password space, however, is not a measure of entropy. An eight-character alphanumeric password has 6.6×10^{15} possible permutations using 95 characters and numbers; though, many of these combinations will not be used in practice [25]. The entropy is a measure of how dispersed and statistically unrelated each character is in the password space. The higher the entropy, the more dispersed passwords are in the password space. Low entropy would result if passwords were statistically dependent events like words or birthdates. As a measure, the password space is often used as a simple metric for the strength of password schemes. In comparison, can graphical passwords equal the theoretical password space of an eight-character alphanumeric password? When considering cognometric passwords, like Déjà vu, a database of images determines the theoretical password space. Déjà vu used 5 indistinct shapes out of 20 distracter images for a password space of 15,504 combinations during a login. Déjà vu had 10,000 images in a database that would be randomly used as distracter images for the 20-image login display. By using a database of 10,000 distracter images, Déjà vu posited that the password space was considerably more than an 8-character password; however, only 20 images were shown at a time with the password images [36]. When considering the password space computation of discrete images in cognometric schemes, the password space is similar to text-based passwords. However, in locimetric schemes, the password space depends on images size and clickable regions. As can be seen from Table 14, a 1024x752 pixel image, divided into 1,925 clickable regions, can have a password space of 2.6×10^{16}. This demonstrates that a locimetric password scheme can have a higher theoretical password space than an eight-character alphanumeric password with possibly less mental effort [40]. With cognometric password schemes, the password space depends largely on the image database containing a large selection of dissimilar images. For locimetric schemes, the size of the image and the clickable accuracy determine the possible clickable regions and hence the password space.

Table 14: Comparison of password space for alphanumeric passwords and PassPoints with different parameters
(Image size and square size are in pixels)

Type	**Image Size**	**Square Size**	**Alphabet Size\ # Squares**	**Text Length\ # Clicks**	**Password Space** (alphabet/squares raised to the power of text length/clicks)
Text	N/A	N/A	64	8	2.8×10^{14}
Text	N/A	N/A	72	8	7.2×10^{14}
Text	N/A	N/A	96	8	7.2×10^{15}
Graphical	451x331	20x20	373	5	7.2×10^{12}
Graphical	1024x752	20x20	1925	5	2.6×10^{16}
Graphical	1024x752	14x14	3928	5	9.3×10^{17}
Graphical	1024x752	14x14	1964	5	2.9×10^{16}

However, another very important attribute of graphical passwords compared to alphanumeric passwords is the entropy of the image itself. For alphanumeric passwords, the entropy is determined by the randomness of the characters in a sequence. If users are allowed to choose words or dates, for example, the entropy is quite low. For images, there are two possible ways to increase entropy. For cognometric passwords, the image sequence should be composed of unrelated images. It might be much harder for users to pick a sequence of images where each image is related to the next compared to picking unrelated characters in an alphanumeric password. For locimetric schemes, the entropy depends on points of interest on the image. Additionally, the image itself, in a set of all possible images, can be a form of entropy. For instance, if users could chose any personal image, the sample space of images would be the population of all possible images a user might chose, which would be a very large set. If images can be restricted to unique occurrences in a population of users, then it follows that it is possible to compute an image hash that would be unique in a set of images. The image hash would have a high degree of entropy if images could be any image a user wanted to choose, such as a family photo. Using a unique hash value, the image hash could be used as part of an authentication scheme [43-49].

Using a multi-resolution representation of signals based on the theory of wavelets, an image could be separated into higher and lower frequency spectrums using band-pass filters [42]. As data integrity and verification of image authenticity became a concern, it became necessary to protect image theft by creating a digital signature based on compressing the low-pass region of image data, hashing it, and possibly encrypting the result. The important step is the low band-pass filter that extracts image features robustly enough to keep those features during compression and decompression [43, 44]. Because of the high degree of information encoded in an image, authentication schemes based around using an image as the input mechanism seem plausible. One design proposed is illustrated in Figure 10 [49]. The image is divided into a P x P blocks and permutated with a secret key, put through wavelet decomposition and filtered for the low-pass coefficients, hashed and then encrypted. The encrypted hash is verified by the authenticating server.

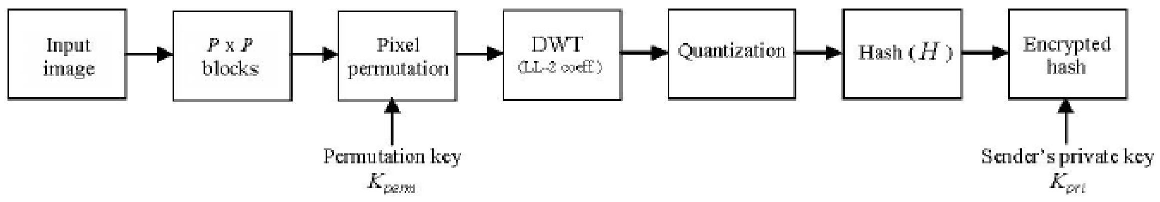


Figure 10: Example of an image hash process used in authentication (Ahmed et al.) [49].

2.8 Graphical Password Weaknesses

Though graphical passwords may show an improvement in recall, there are some weaknesses that can be exploited. Any scheme relies on the entropy of the unique identifier that a password produces. The more random and variable a password, the harder it is to guess. Weaknesses can occur if the image or images lead to easy guesses, shoulder surfing, or practical implementation problems. Indeed, graphical passwords can suffer from some of the same weaknesses that plague alphanumeric passwords.

With cognometric graphical passwords, a sequence of images, and the quantity of the distracter images selected from a large database of discrete images, forms the entropy of the scheme. With locimetric schemes, a database is needed with many natural scenes that produce memorable, selectable areas. This implies that cognometric and locimetric passwords need a large database of dissimilar images. In short, the image database helps determine the password strength [36-40]. There are several practical problems with these schemes:

- It could be difficult to find a large selection of images that do not infringe on copyright law;
- Vetting images to find memorable ones could also be a difficult task.

In both cases, a database of thousands of images could seem daunting or impractical. An alphanumeric password authentication scheme only relies on a keyboard of letters, numbers, and special characters. It might be understandable that Web administrators can implement text-based authentication schemes easier than one that needs a large database of images.

Another problem is the extended login time and effort with some graphical passwords. Passfaces™, for example, needs 16 to 17 screens, of 9 images on each screen, to have the comparable strength as an eight-character alphanumeric password [37, 50]. This would present a long series of screens to the user and increase the login time when compared to input from a keyboard. De Angeli et al. demonstrated that a four-image cognometric scheme cost the user considerable time compared to a four-digit PIN (Figure 11) [39]. A study with PassPoints showed a similar increase [40]. Table 15 shows that during practice and successful logins, there was an increase in login time compared to alphanumeric passwords. However, thinking of a hard-to-guess alphanumeric password may have made the creation time higher for alphanumeric passwords.

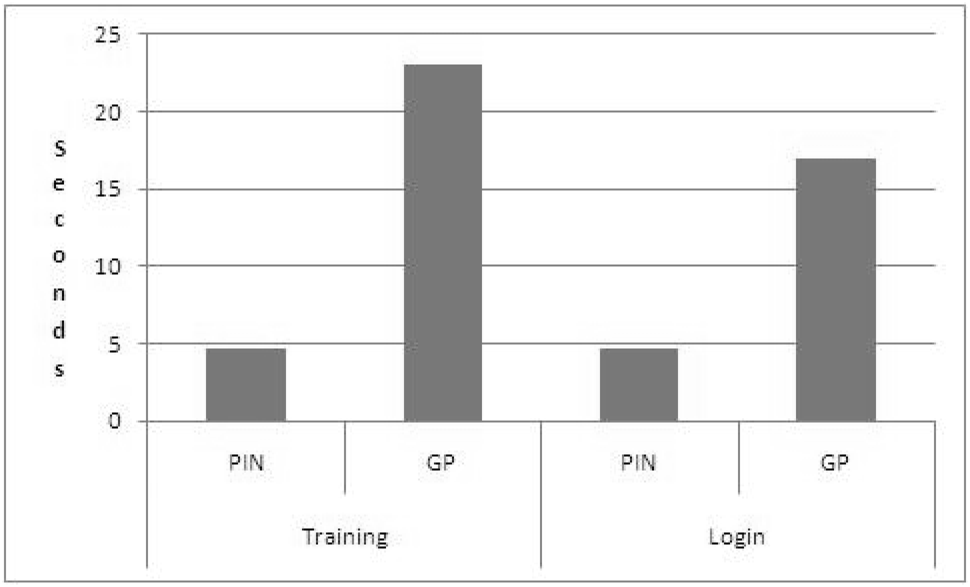


Figure 11: Training and login times for PIN vs. graphical passwords (GP) (De Angeli et al.). [39].

Table 15: Time in seconds of PassPoints versus alphanumeric passwords [40].

Metric	**Mode**	**Mean/ SD**
Total time to create	Alphanumeric	81.10/36.50
	PassPoints	64.03/21.93

Total time to practice	Alphanumeric	66.08/4.92
	PassPoints	171.89/24.46
Time for correct submission (1st week)	Alphanumeric	5.23/1.66
	PassPoints	8.78/4.40
Time for correct submission (6th week)	Alphanumeric	9.24 /3.72
	PassPoints	19.38 /17.57

The problem with an increased login time is the increased chance for shoulder-surfing. A study by Tari et al. compared the susceptibility of Passfaces™ to shoulder surfing when compared to strong alphanumeric passwords [51]. Experimenters acting as attackers were allowed optimum positioning and to take notes. Passfaces™ requires five pictures to be chosen so a 5-character password was required in the comparison. Two types of text-based passwords were used: one that could easily be attacked by a dictionary attack and one that was very hard to guess. The results showed that a graphical password could be more resistant to shoulder surfing when compared to easy to guess text-based passwords. However, Passfaces™ was fifteen percent easier to shoulder-surf than non-dictionary passwords but three times harder than dictionary passwords (Table 16). This could suggest that the movement from screen to screen, when choosing the correct sequence of images, increases the chance of shoulder surfing because of an increase in input duration.

Table 16: Average correct, ordered entries of a shoulder surfing attacker [51].

Authentication Type	**Average**	**Std. Deviation**
Non-dictionary password	3.65	1.631
Passfaces™ with a mouse	3.10	1.119
Dictionary password	1.30	0.923

An area of locimetric weakness, not relevant in alphanumeric or cognometric password schemes, is the effect of guessable regions called hot-spots. Chiasson et al. proposed a persuasive method that limits predictable choices when clicking on an image [53].

Similar to producing stronger alphanumeric passwords using an application that will monitor and persuade users to enter a more entropic password, images can also enforce random choices [53, 54]. If no persuasive measure is taken to ensure that users pick random points, users may pick areas that provide a stronger cued recall. Thorpe and Van Oorschot conducted a study of 17 different click-based images. Van Oorschot et al. noted that regions of possible guesses would involve the precision of the click point, or the area of tolerance, and if the object is distinguishable from its surroundings such as a white spot on a black hat [55]. Using cluster analysis on an image size of 451 × 331 pixels and a 19×19 pixel square of error tolerance, users clicked on five separate points. A cluster was defined as one or more clicks in a 19x19 pixel tolerance (Figure 12). The Van Oorschot et al. study found a strategy that could guess 36% of passwords with a 31-bit dictionary. The study experimented with an automated image dictionary attack that guessed 22% of 28-bit dictionaries.

The Van Oorschot study illustrates that locimetric schemes can suffer from decreased password space and automated dictionary attacks due to areas of common interest. Because an image can have hot-spots, the size of the images needs to be one that allows for more clickable regions. The smaller the image size, the higher the probability that hot-spots will converge onto a guessable password. Some entry screens may not accommodate a large image size whereas alphanumeric passwords require little space to input a password. Additionally, images must be chosen that have a high degree of possible objects so users will not converge on a few clickable areas.

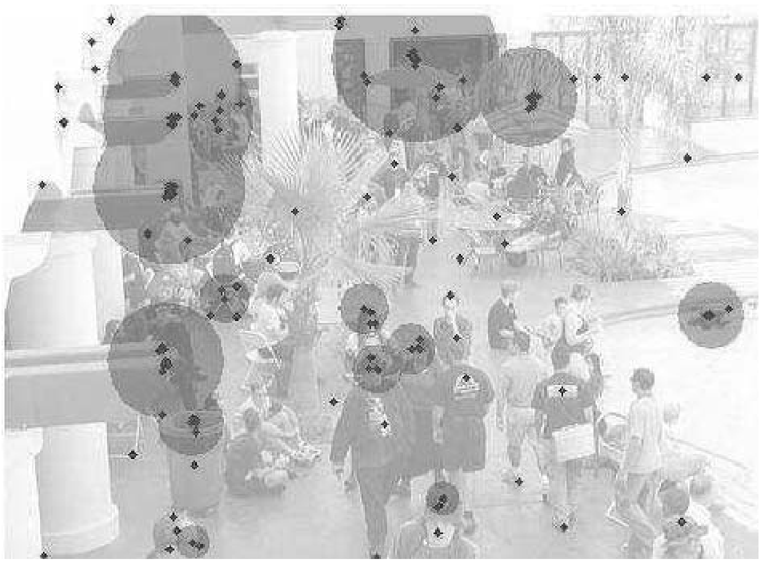


Figure 12: Hot-spot clusters indicated on an image (Van Oorschot et al.) [55].

2.9 Graphical Passwords and Anti-Phishing

One area where graphical passwords are very useful is in the prevention of phishing attacks due to key and mouse logging Trojans. Doja and Kumar tested a locimetric image that had discrete 20x20 pixel clickable areas with an image size of 451×331 pixels. They proposed that the sequence of clicked squares could be added to a 128-bit MD5 hash to produce the resultant password. Though this study did not test their solution against spyware, they argued that since locimetric schemes did not use input from the keyboard that these schemes would be resistant to key-logging attacks. Likewise, they noted that mouse movements would also not predict anything meaningful because the image can be in different screen locations that would confuse mouse logging spyware [56].

2.10 Some Common Hardware Authentication Schemes

Text-based and graphical passwords satisfy one of three ways to identify users: what you know. However, authentication schemes can incorporate two more: who you are and what you have. A system that records what you know can be vulnerable to dictionary attacks or spyware. However, if a password system can use something physical as part of the authentication, it arguably would be harder to guess or capture by Trojan programs. A biometric device measures some unique human feature. Common biometric devices can measure a user's fingerprints, facial structure, retina, voice, and hand structure, to name a few. Some commercial applications can incorporate these devices into a special mouse, keyboard, or as separate capturing devices [57]. Biometric devices are getting small enough for mobile phones and the cost is decreasing.

Though many government agencies and consumers have embraced biometric devices, a current UNISYS survey found that the financial industry has been reluctant to use them as authentication schemes because of negative consumer opinion over privacy, less mature technology, costly implementation, complex installation, cheaper alternatives, and high consumer cost [58]. At the time of writing, a Microsoft Fingerprint Reader was found new on Amazon for $130 [59]. A commercial survey of 1,396 adults by AuthenTec in 2004 found that only 23% of users would pay over $50 for a biometric device with 43% willing to use a fingerprint in lieu of a password [60]. A survey of biometric companies found that 35% of potential customers are utterly confused on what to buy and 36% were not ready to buy any devices at all [61]. As costs continue to decrease, devices become smaller, and industry standards for implementation and deployment are better standardized, biometric devices would offer an effective authentication system for consumers.

However, biometric devices are not immune to error. O’Gorman compared passwords, tokens and biometric devices to illustrate security strengths and weaknesses [62]. O’Gorman described two types of biometric error: FMR and FNMR. When the device gives a false match, it is measured by a false match rate (FMR); that is, the signal the device outputs is the same as another user. If the device gives a false rejection when it should have accepted, it is a false no-match measured as false no-match rate (FNMR). A high FNMR is common with biometric devices because of environmental conditions such as low light, improperly placed fingers, improper use of the device, or an unstable device signal. Because of the environment, the user’s knowledge, or the device quality, biometric devices can reject a user repeatedly. O’Gorman surveyed several sources and collected FNMR and FMR statistics for various biometric devices (Table 17). Biometric facial scans had a false rejection rate of 16%, as a worse case, and a 16% false acceptance rate. The iris scan had the best false rejection and false acceptance rate of all biometric devices in Table 17. Biometric devices may offer a secure way to authenticate users because of unique signal generation and their resistance to spyware; however, biometric devices can have a high FNMR, are often difficult to use correctly, and have a cost point that many users find too high.

Table 17: Biometric recognition error rates (O'Gorman) [62].

Biometric	**Attempts**	**FNMR**	**FMR**
Face	1	16%	16%
(1-3 mo. Spaced)	3	6%	6%
Fingerprint	1	2%	0.02%
	3	2%	0.01%
Hand	1	3%	0.3%
	3	1%	0.15%
Iris	1	2%	0.0001%
	3	0.25%	0.0001%
Voice	1	7%	7%
(text dependent)	3	2%	0.03%

There are cheaper hardware authentication schemes that are resistant to spyware attacks and easier to use. O'Gorman noted that a hardware token with a password would be quite secure. A token can be a physical device, often removable, such as a USB key or smart card that aids in authentication by securely storing passwords or a unique identifier [63]. Sometimes the word token is used to mean an identifier issued from an authenticating server as a software token [64]. Hardware tokens are a recommended approach for two-factor authentication with financial institutions according to the guidelines of the Federal Financial Institutions Examination Council (FFIEC) in the United States [65]. Some banks authenticate with a debit card number and password [66]. Presumably, this authentication scheme is considered two-factor: what you have and what you know. However, having a card with the number visibly printed on the front is a weak form of two-factor security. By contrast, a smart card can keep a unique number hidden on a strip but needs a card reader [67]. As with biometric devices, this requires the user to buy an additional device as either a separate hardware unit or as part of some special keyboards.

USB hardware tokens, on the other hand, may be a better alternative in two-factor authentication because of cost, usability, and portability. USB tokens can contain a tamper-proof, cheap microprocessor that encrypts or aids in secure authentication, along with a flash memory that maintains a user state. In two-factor authentication methods, a

USB hardware token with a password would satisfy what you know and what you have [68, 69]. Because the USB token is small and portable, security also involves physical possession of the device separate from a PC. However, many of these solutions suffer from the same weakness as a password vault mentioned previously: namely, a keyboard is needed to enter the main password. Though more difficult, a hacker could log the password with a key-logger while the user enters the device and then steal the device to have all the passwords. Once access is granted into the device, the attacker can access any Web site since the device automatically logs the user in. Therefore, with these solutions, the main security feature is the physical possession of the device. Though these solutions propose to either use constantly changing passwords [68] or passwords encrypted by the Advance Encryption Standard (AES) in a tamper-proof system [69], the fact remains that the device holds all the passwords, and as such, are a high-value target.

Attackers not only can steal the main text-based password that allows entry into the USB token with a simple key-logger, these devices are also vulnerable to implementation attacks such as side-channel, fault, and probing attacks [70]. Because a device that holds passwords would be a high-value target, attackers could buy the product and look for implementation weaknesses to attack. One example, demonstrated by De Mulder et al., found that a popular FPGA public-key cryptosystem circuit leaked information through electromagnetic radiation [11]. Using a loop antenna surrounding the chip, both simple electromagnetic attacks (SEMA) and differential electromagnetic attacks (DEMA) where analyzed. The sudden change in current that occurs when a CMOS gate goes through a voltage transition causes magnetic flux over the chip, which induces a voltage in the loop antenna. The resultant voltage can be analyzed by statistical analysis. By changing the input to the microprocessor and monitoring the induced voltage signal, the RSA key was extracted after only 1000 measurements. Though this is one example on one kind of hardware system, there exist several ways hardware can be attacked. Once the attacker finds a weakness, it could be lucrative to send thieves out to steal these devices and bring them to the lab. This cannot happen with biometric devices where the actual user must be present; however, given the fact that USB tokens need no installed drivers, are small, can be carried in one’s pocket, are cheaper than biometric devices, then this might be an acceptable tradeoff.

CHAPTER 3 PROPOSED SYSTEM OVERVIEW

Due to the prevalence of key-logging spyware and the difficulty users have with creating high-entropy, text-based passwords, a novel graphical password, token-based scheme is proposed. Given that spyware can masquerade as legitimate software applications or infect other applications, a read-only hardware system is necessary. A token-based USB key system using secure file I/O offers such a solution. Most desktop and laptop computers using various operating systems support the USB port; therefore, no special hardware is needed to support USB token keys (alternatively, a Bluetooth based device could be constructed for smart phones, etc.). Additionally, by using file I/O, most operating systems will not require the user to install any drivers or configure any settings, implying that the installation of the device requires no user involvement. This thesis proposes a novel solution that combines the strength of graphical passwords with a low-cost USB key hardware system.

Many weaknesses of graphical passwords have been explored. From these weaknesses a better click-based graphical password has been designed to eliminate hot-spots, guessing, mouse-logging, screen capturing, and usability issues such as click-accuracy. By allowing the user to choose meaningful, family photos, or other images that they prefer, the issue of having a large database of copyright free images is resolved. Superimposed on the image, are randomly placed letters and numbers to form a virtual keypad. This virtual keypad reduces the problem of users remembering the exact click-spot when they recall their password, as well as preventing key-logging spyware. The randomness of the x-y positions of letters and numbers on the underlying image reduces hot-spots. Additionally, the underlying image helps provide cued recall of the clicked password. As part of the implementation, persuasion is used to further randomize what a user can select for letters and numbers when creating a password. By requiring a family photo, or other desirable images, the probability of two users having the same photo is infinitesimal. The entropy of the image itself with the randomized virtual keypad superimposed on top will help form a cryptographic hash. The cryptographic hash will be used as either a 256-bit AES key for another secure layer of encryption more resistant to man-in-the-middle attacks or a 32-character alphanumeric password.

The danger of Trojans and other viruses is their ability to attack PC memory, data, or applications that can possibly lead to password theft. The proposed solution will use an embedded Web browser on a read-only USB device. With a read-only device, it is impossible to infect the firmware or flash memory. As far as the user is concerned, the USB token device looks like a regular file system. The user will access the USB device like a normal file system and run a Web browser by double clicking on the executable program. When the Web browser is loaded from the USB device, a novel approach is used to validate that the binary image of the executable has not been infected or tampered with by malicious software. All data transferred between the USB device and the user's PC is by file I/O using 256-bit AES encryption. The encryption of data will depend on two graphical passwords. One graphical password will authenticate the user to the device and encrypt or decrypt data between the device and the main application. The other graphical password will be used to create an AES key to encrypt Web data or form a stretched text password. The images will also help identify the device to the user, so they know if they are using the correct device before entering their password. The following diagram provides an overview of the operation of the system.

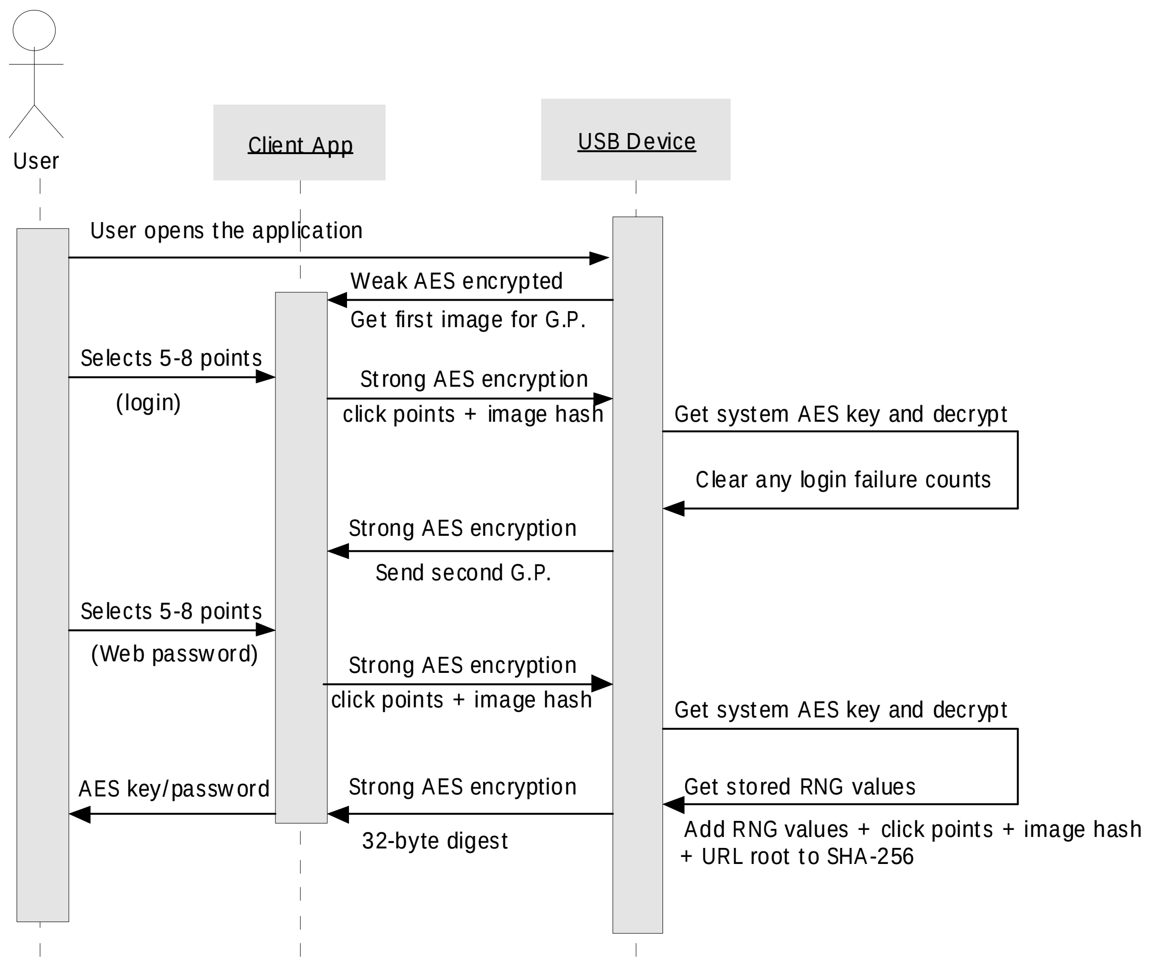


Figure 13: Sequence diagram of proposed system (G.P. =Graphical Password)

CHAPTER 4 SOFTWARE SYSTEM

The client application incorporates two graphical passwords into a secure web browser to make authentication simple and fast. The first graphical password is used to log the user into the USB device. The second graphical password helps the USB device return either an AES key or a long text password for Web authentication. The Web browser also assists the user in initiating a secure login. That is, the final goal of the client Web application is to provide a secure password which is more resistant to Trojans than other available solutions. To provide this security, the main program that the client uses is loaded from a read-only USB flash device. The USB device is controlled by a microcontroller that can verify and alter the data sent to the client system. One important module in the client system, called the Device Manager, handles communications with the USB device. The Device Manager uses a secure way to transmit data on a USB bus using simple file input and output (I/O). Since most operating systems support a USB port and block file I/O with a storage device, the user need not install any drivers for the device. An overview of the system is illustrated in Figure 14.

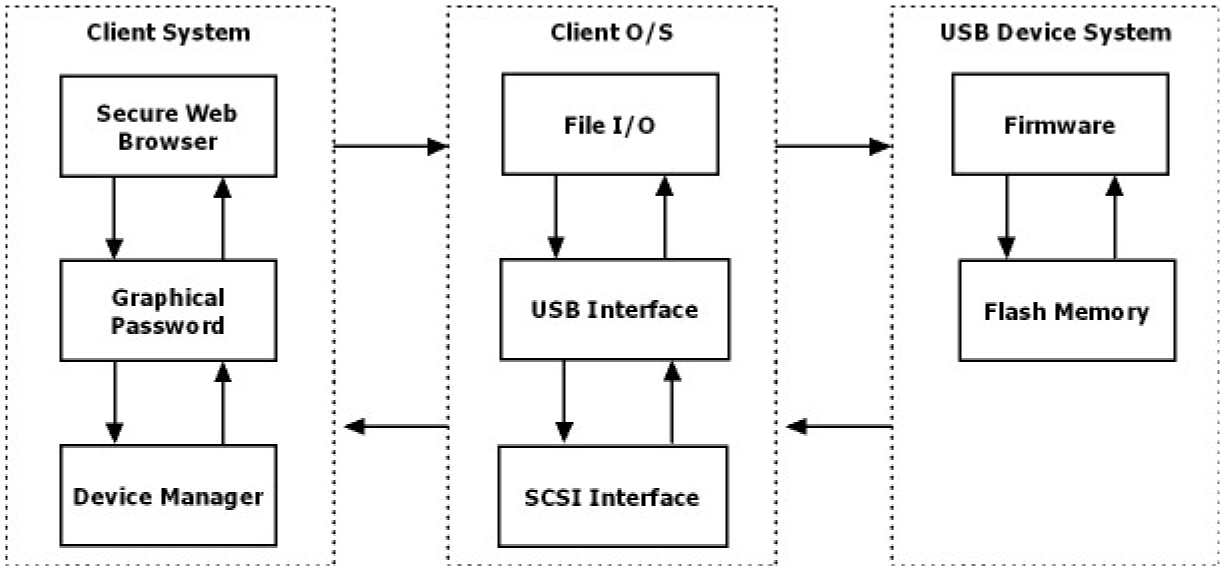


Figure 14: System Block Diagram.

4.1 Graphical Password Application

Because Trojan key-loggers are a real and significant problem, and because users often do not select long, random alphanumeric passwords, graphical passwords seem to offer a better solution. Graphical passwords can prevent key-loggers, mouse-loggers, and image capturing while the user inputs a password. Therefore, threats from spyware are reduced or eliminated. If users do not need a keyboard, key logging is impossible. However, mouse logging and screen capturing might be possible. The proposed solution will demonstrate that these are not a threat. Additionally, it has been shown that alphanumeric passwords are vulnerable to dictionary attacks because users often pick low entropy passwords [25]. By using a graphical password, the proposed solution will demonstrate that a 32-character, random, password can be created from 5 to 8 mouse clicks on an image. See Figure 15 for an example.

Wiedenbeck et al. showed that click-based, or locimetric, password schemes, such as PassPoints, had a lower login error rate over time compared to alphanumeric passwords [40]. However, PassPoints suffered from an extended learning phase and increased login time when compared to alphanumeric equivalents. The main difficultly in PassPoints was that users were not able to accurately recognize their click points. In the extreme case, if a user had to find their click-point on a completely white screen, it would be hard, if not impossible. At the other extreme, if the image has a lot of clutter or repeating objects, the user would find it difficult to remember the correct location. Therefore, the recall and mouse-click accuracy seem to be highly dependent on the image type [41]. However, the proposed scheme hopes to overcome the problems found in locimetric schemes while leveraging their strengths.

4.1.1 The Proposed Graphical Password Scheme

The proposed scheme is a locimetric scheme that uses an image of the user's choice. Because the image seems to be the main reason for varying login success and mouse-click accuracy, this scheme will moderate the importance of the image and provide randomly located, well demarcated click-points that the user can easily recognize. By using well demarcated areas, the user will not have to guess the tolerance and suffer from mouse-click inaccuracy. Additionally, in each demarcated area, a random letter or number will be placed. Because users are transitioning from a keyboard and alphanumeric passwords, putting letters and numbers over the image is likely to help with recall, especially if mnemonics are used. The image underneath the text will help aid in cued recall; however, since the user can also remember the text, the image does not have to be restricted to ones that are the most effective. Additionally, by allowing the user to choose a personal photo, the recall of their password may be enhanced. Therefore, the system proposed is a hybrid of an alphanumeric password and a graphical password. Using the same image-tolerance as PassPoints, the image is restricted to 451x331 with a click-point radius of 10 pixels. A circle is used to demarcate the allowed clickable location to the user. The user also has the ability to change the circle color or text color to allow for better contrast against the image. Additionally, the image intensity can be reduced, on a graduated scale, so the user can see the text better if required (Figure 15).

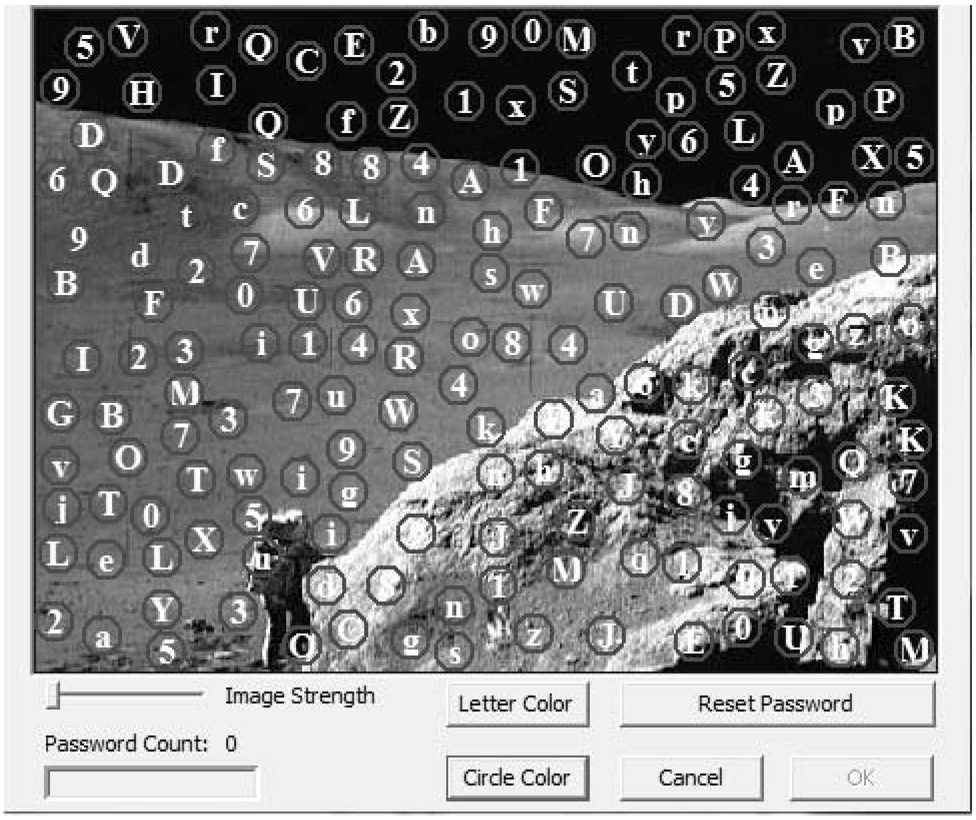


Figure 15: Screen capture of the hybrid alphanumeric-locimetric password scheme.

4.1.2 Password Space

The password space is determined by how many click-points are allowed on an image. Additionally, the number of points should be a multiple of 4, as will be discussed. Given the image size, it was found that 180 points, with a 10 pixel radius, provided the optimal fit. The user will be required to pick at least 5 points resulting in a password space of: 180^5 or $1.9\text{x}10^{11}$. However, if the user were required to pick as many as an 8-character alphanumeric password, the password space would increase to 180^8 or $1.1\text{x}10^{18}$ possible choices. In contrast, an 8-character alphanumeric password only has $6.6\text{x}10^{15}$ possible choices using 95 characters. The worst case scenario is that the user will have to memorize a random sequence of 5 or 8 characters just as with an alphanumeric scheme;

however, based on research of locimetric schemes like PassPoints, the picture superiority effect and cued recall can assist the user to memorize a random password.

4.1.3 Password Communication Resistance

A major problem, as outlined by Sasse and Adams [15], is that users might write down their password instead of memorizing it. Additionally, the user might verbally communicate their password when they should not. To help prevent users from scribbling down the text that comprises their password, a multiple of each character is placed on the graphical password. First, a random generator picks a subset of upper or lowercase characters and single digit numbers to form a list of 45 characters. From this list, each character is replicated to give 4 copies of each character. Even if the user writes down their password text, there are 4 of copies of each symbol; and hence the password is still not uniquely defined. If the user's password were known to an attacker simply by the text, and given that the password were to be 8 characters long, there would still be a password space of 4^8 or 65,536 possible choices. If screen capturing is disabled, the user would have to manually sketch the password text and their locations. Though this will not prevent a user from communicating their password, it does make it more difficult when compared to alphanumeric passwords.

4.1.4 Hot-Spot Resistance

As demonstrated by Chiasson and Van Oorschot et al. [53, 55], users will tend to pick points that are easy to remember. The type of image highly influences this behavior. For example, in Figure 15, the text around the astronaut is easy to find and remember compared to the black space above the moon. An attacker might suspect that the astronaut is a region of interest when guessing the user's password. These hot-spots can be eliminated if the user is persuaded to use more random locations. To accomplish this persuasion, the client system will darken and disable random click-spots while the user is setting up their password. Some random points may be near hot-spot areas but not all of them. Heuristic image analysis could be automated to further restrict how many random points are allowed in image areas with certain features. Additionally, adjacent text on top of the image could form a hot-spot. For example, adjacent letters might form the word "cat". It would be useful, therefore, to have a persuasive method that can look for dictionary words with adjacent text. For the proposed solution, only half of the points on the image will be allowed as a password choice. Those areas that are allowed are randomly chosen and do not account for image features or adjacent text.

4.2 Graphical Password Generation

As was demonstrated, the password space resulting from the choice of 8 out of 180 items, randomly placed on an image, is larger than an 8-character alphanumeric password. However, there is more information that can be encoded with each click-point that creates an even higher source of entropy. Unlike alphanumeric passwords, which only have 95 characters as a source of entropy, an image has a large source of features that can be captured by decomposition techniques [42-49]. The image, under the text where the user clicks, will contain pixels of various colors. Additionally, the whole graphical password with the random text, their random positions, and the image can be unique among a large population of users, especially if users provide their own image. Using these collective factors of variability in a graphical password, it will be possible to stretch a 5 to 8 point selection to a longer password. The goal is to transform the entropy of the graphical password and chosen click-points to a 256-bit, AES key or long series of text suitable of most Web login pages. Using 256 bits gives a theoretical password space of 2^{256} or $1.16\text{x}10^{77}$ possible combinations. Figure 16 shows a block diagram of the process.

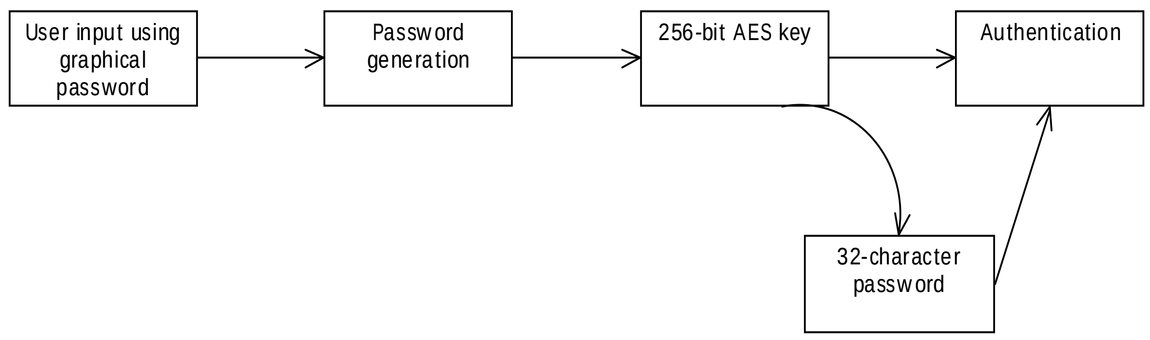


Figure 16: Block diagram of graphical password input to final product.

4.2.1 Cryptographic Hash

To accomplish a collision-free AES key, a cryptographic hash function, $H^*(m)$, offers a realistic implementation mechanism given as: $H^*(m) = P_{key}^{256}$ where P is the 256-bit password (message digest) and *m* represents the graphical password input that comprises the message. However, this assumes that the combination of what the user selects as a password, and the graphical password image, can give an input message, *m*, with no collisions. That is, a cryptographic hash function requires that the input message be unique if the resultant message digest, P_{key}^{256}, is expected to be unique: a one-to-one relationship. Unique message digests derived from all users of the graphical password system should be resistant against guessing. Additionally, a property of the cryptographic hash function is to make it hard to find the message, *m*, given the hash function, and message digest; therefore, the user's graphical password click-points and other components of the message input are hard to attack. This would allow the user to safely keep the same graphical password image and click-points even if the message digest were compromised. In summary, two attributes of a cryptographic hash function that are useful [71]:

• It is extremely difficult to reconstruct the input data from the output.

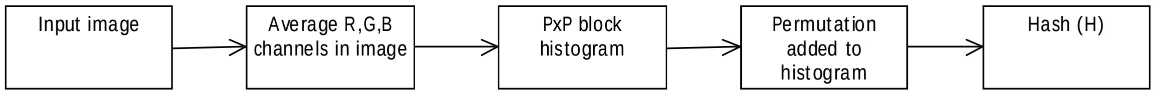
• It is extremely unlikely that the hash function will produce the same output given different inputs.

One such cryptographic hash function is the Secure Hash Algorithm (SHA) 256 which is compliant with the FIPS Publication 180-2 specification [72]. The SHA-256 hash function will give a 256-bit message digest that can be used as a 256-bit AES key.

4.2.2 Histogram Image Hash Algorithm

To provide a unique input message to the cryptographic hash function, the uniqueness of the user's family photo will be used to derive a unique image hash. The input to the cryptographic hash function follows: $H^*(H(image) + CP) = P_{key}^{256}$ where CP is a stream of click-point data from the graphical password and *H* is the image hash function. The image hash function will offer the largest contribution to uniqueness of the input message since the click-points only constitute $1.1\text{x}10^{18}$ possible combinations if 8 click-points are selected. An image is composed of pixels which are defined by a red, green, and blue color channel having values from 0 to 255. If each value in the color channel was an independent event with equal weight, the theoretical combinations would be $(256)^{3*p}$ where *p* is the total number of pixels in the image. Given an image composed of 451x331 pixels, that is 256^{447843}. With such a large number of theoretical combinations, it is safe to assume that if every user picked their own family photo, the possible permutations approach a very large set. However, with this system, the image is too large to form a message into the cryptographic hash function; therefore, a reduction to a smaller hash value is necessary.

The image hash function will use a histogram composed of counts that represent how many color value averages in a P x P pixel block that are above a certain color threshold of any red, green, or blue color channel. This is a simple yet effective low-pass filter. Added to each histogram value is a corresponding permutation. Figure 17 shows a block diagram of the process to generate an image hash.



Input image
Average R,G,B channels in image
PxP block histogram
Permutation added to histogram
Hash (H)

Figure 17: Block diagram of histogram image hash generation.

The graphical password image will be of the dimension W x H pixels and will be partitioned into non-overlapping P x P pixel blocks for a total of W*H/P^2 blocks. The total histogram values will be the integer value of W/P indicated by *L*. In this solution, *L* = 56 histogram values of one byte each. Each color channel will be represented by *R*=red, *G*=green, and *B*=blue. The image threshold for each color channel will be:

$$\overline{T_R} = \frac{1}{N_p}\sum_i^W\sum_j^H R_{ij} \ , \ \overline{T_G} = \frac{1}{N_p}\sum_i^W\sum_j^H G_{ij} \ , \ \overline{T_B} = \frac{1}{N_p}\sum_i^W\sum_j^H B_{ij} \ .$$

The threshold is defined for red, green, and blue respectively, where N_p is the total number of pixels in the image. Each block will have an average RGB component:

$$\overline{B}_{lm} = \frac{1}{PxP}\sum_i^P\sum_j^P C_{ij} \quad .$$

The value C_{ij} is composed of values R, G, and B for each pixel at the ith and jth position with respect to block B_{lm} and l is the index for the block along the length of the image and *m* is the block offset along the height of the image. Each histogram value is computed by:

$$N_l = (\sum_m^n \overline{B}_{lm} \geq \overline{T_R}) \vee (\sum_m^n \overline{B}_{lm} \geq \overline{T_G}) \vee (\sum_m^n \overline{B}_{lm} \geq \overline{T_B}) \, .$$

The value N_l is the count at position *l* of *L*. If any value is larger than 255, the upper limit of a byte, the value will be allowed to roll over. However, the mean color for each channel, as the threshold, keeps the count to a relatively low number compared to 255.

A permutation is added to the final count before moving to the next index in the histogram. The permutation is computed using only the blocks with a color channel that have an average equal or greater than the threshold for blocks 1 to *m* given as:

$$K_l = K_l \otimes (B_{lmR} + B_{lmG} + B_{lmB}) \wedge B_{lm} \geq \overline{T} \ .$$

If the permutation that is added to the count is over 255, the value in position *l* of the histogram will be allowed to rollover. Finally, the histogram image hash is represented as:

$$H(image) = \sum_{l}^{L} N_l + K_l .$$

When the user clicks on the graphical password, in the designated areas, both the text and its location are unique to that graphical password. In addition, the pixels under the text may also be unique, though not necessarily. For each point the user clicks, a sweep of a 10-pixel radius over the image underneath can be used to find the average color. If the user chooses an 8 click-point password, that would result in 8 bytes of ASCII text, 32 bytes of X-Y data (each X-Y value pair contains two words of data) resulting in 40 bytes of data. The average color of the image with a 10-pixel radius about each point adds 8 double words of data or another 32 bytes. Therefore, from an 8-click password there will be 72 bytes of data. If that is added to the image hash calculated above, the result is now 128 bytes of data as the message for the cryptographic hash function. Though it may be possible to have a collision, it seems unlikely with 128 bytes of data elements derived from the uniqueness of both the image and the user's password (Figure 18).

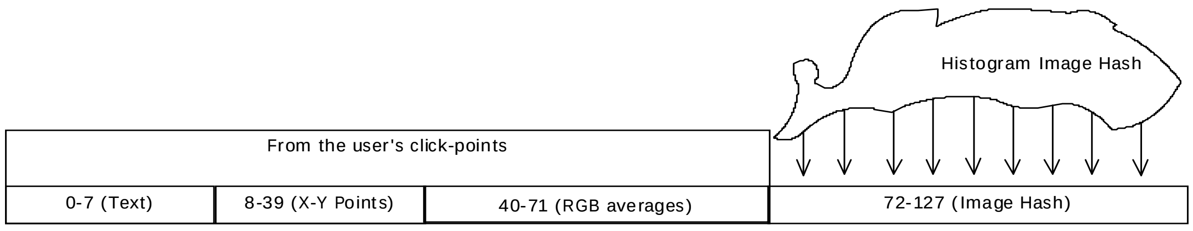


Figure 18: 128-byte message structure for the cryptographic hash function.

4.2.3 Cryptographic Hashing and the Device

For security, it is important that the user create their AES key using the USB device. This will ensure that only the owner of the device can create the correct AES key or 32-character text password. Therefore, true, not pseudo, random values stored in the device will be added to the input of the cryptographic hash function before it returns to the message digest to the client application. The message digest is represented by:
$H^{*}(H(image) + CP + RNG) = P_{key}^{256}$.

Function *H(image)* produces the histogram image hash and *CP* is the click-point data that both come from the client application. The device provides RNG values to the cryptographic hash function which both reside in the device.

4.2.4 Text Password Generation

The text-based password scheme currently in use on the Web today would be impractical to replace anytime soon. The weakness of the text-based password scheme is not only because of key-logging and clipboard directed spyware, but the fact that users tend to use easy-to-guess passwords. Graphical passwords have typically been used to work with special infrastructure, such as Passfaces™ [37]. Using the message digest from the cryptographic hash function in the device, 32 bytes containing values from 0 to 255 can be mapped to text characters. Although ASCII ranges from 0 to 127 and extended ASCII up to 255, authentication applications may allow only 26 upper and lowercase text, single digits and the common special characters found on most keyboards. In some instances, the single quote, double quote, and back-tick are not allowed because they are used in some SQL and script injection attacks. If the values in the 32 byte (256-bit) message digest are random and unique across a large population of users, then the values in each of the bytes can be considered a good source for choosing text from a set of allowed characters. Using each byte in the message digest, a mapping function can be derived such that $P_{key}^{256}(i) = \{allowed\ ASCII\ set\}$ where P is the cryptographic digest function and the value at index, i, is mapped to allowed ASCII text. It might be possible to use most of the ASCII and extended ASCII set; however, if leaving off a few possible blacklisted characters and using only 90 items, that would leave 90^{32} or 3.4×10^{62} theoretical combinations. Putting that in perspective, if all of the values, ranging from 0 to 255, could be used then the password space would have 1.16×10^{77} combinations. Using less than half of the range reduces the password space significantly but still it is much higher than an 8-character alphanumeric password and it is derived from random, unique values.

4.3 Screen and Mouse Capture Prevention

Graphical passwords can be attacked by spyware that targets the image of the graphical password and mouse activity. Each operating system will have various low-level system calls that can both enable and prevent these attacks. For example, the proposed system works with Microsoft Windows and, as such, will use APIs specific to Windows. In Windows processes, Dynamic Link Libraries (DLLs) can "hook" into window messages on a local or global level. This has security issues in that spyware can hook into the mouse movements or key presses of any process on the system. A malicious application might call the SetWindowHook API to log mouse messages [85]. Additionally, a screen capture of the graphical password dialog could be acquired by an attacker. The attacker could relate screen locations and mouse activity to the graphical password window. The system proposed uses DLL injection to take control away from some of these message hooks while the graphical password is displayed. Any Windows APIs, message hooking techniques, or other libraries that are purported to function according to the documentation, are subject to change and will always need careful evaluation. For this reason, any implementation is ephemeral at best. Constant vigilance and updating the client application for a particular operating system is necessary.

4.4 Secure Web Browser Application

It is not enough to provide only a graphical password to the user. A graphical password must work with the current infrastructure in a secure manner and provide a more secure solution than current text-based password schemes. Therefore, the proposed solution provides a secure Web browser with the following attributes (Figure 19):

- The graphical password is integrated into the Web browser.
- The application helps the user set up a device login graphical password.
- The application provides a login screen to the device.
- A graphical password is provided that can be used to generate a text password.
- A text password is automatically inserted into the password field without the keyboard, clipboard, or other easily attacked methods.
- The application sets up an AES encrypted communication channel with supporting Web servers instead of weaker PKI cryptography.
- The application uses frequent AES key exchanges with supporting Web servers.
- Uses stored links in the device as favorite links so the user will not have to type URLs.
- Inform the user if the client has been attacked by a virus and abort the application.

The current prototyped solution uses a Web browser ActiveX control in a container built with Visual C++. It should be noted here that there is a danger using outside libraries installed on the user's PC since they can be attacked by malware. As mentioned in the section under malware countermeasures, the ActiveX control would have to be vigilantly checked by the container before the application could proceed. An open source browser like Firefox would offer fewer library dependencies. Whether an ActiveX control or an open source browser is used, there needs to be a container that manages the Web session and interacts with the USB device. The container would not have to be responsible for the HTML document implementation. Rather, the container can interact with a prebuilt Web browser like an ActiveX control, where all document objects in an HTML page are accessible programmatically. That is, as with JavaScript, any form, field or other HTML object can be manipulated by the container.

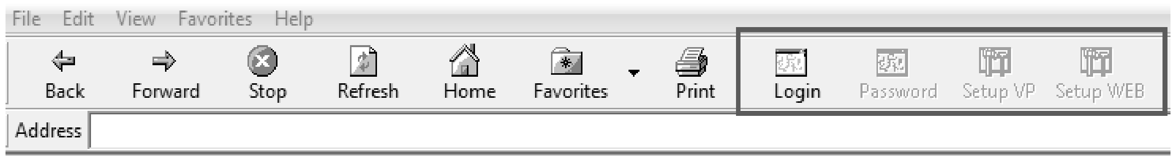


Figure 19: Screen shot of the embedded browser’s toolbar with graphical password buttons.

4.4.1 Device Setup

The browser’s toolbar has a menu for the graphical passwords and setup utilities (Figure 19). The user should not have to exert any special effort to input a password. Clicking on the toolbar to activate the graphical password is all that is required. The user will initially receive a USB device from some commercial outlet configured with a default password. Since a graphical password is hard to communicate, unlike a PIN, the user will be given a pseudo-PIN issued with the device. The pseudo-PIN will be any five letters or numbers on the graphical password screen without regard to position or case sensitivity. After the user enters their pseudo-PIN on the graphical password, the user will need to find two images that they like. One image will be used for the login to the USB device and the other for their Web passwords. The image, as mentioned before, can help aid in cued recall. However, it also serves as a security feature. Since the user will pick an image they prefer, if someone switched their device, the user would know it before entering their password. Once the image is configured, the user can press a button to randomly generate the text and associated locations over the image. When ready, the user will select a password with the appropriate minimum requirement. In contrast, the graphical password for the Web is never setup (except the image). No Web passwords are stored in the device for security reasons; therefore, the user must always enter their Web password every time as needed.

4.4.2 Text Password Insertion

After logging into the USB device successfully from inside the Web browser container, the user can select or type the URL to a login Web page. The user will activate the Web graphical password within the browser and click their password. Once the Web browser receives the device-generated message digest and forms a text password, an obvious indication will notify the user that the password is now ready for insertion into the password field. The user will then just click the mouse pointer on the password field and the underlying application will find the HTML document object and programmatically insert the password into the password field. Any buffers holding the password are inside the client process and are not accessible to spyware.

4.4.3 One Graphical Password-Multiple Text Passwords

One problem with text-based passwords is the cognitive load on users to maintain many different passwords and constantly remember new ones. For better security, many administrators will require a password to expire. With many passwords to remember on a regular basis, users tend to write them down and put them in insecure locations [15]. With the solution proposed, using the security of the USB device and the cryptographic hash function, a user will be able to generate many long, random and dissimilar text passwords from the same graphical password. In addition, for security, there is no need to store the user’s password in the USB device to accomplish this.

As previously mentioned, both the click-point information of the graphical password and the image hash are used as one part of the input to a SHA-256 cryptographic hash function. Additionally, random values in flash memory are added as the device’s contribution into the hash function which results in a 32-byte (256-bit) digest. True random (RNG) values that are added in the device are an important part because it enforces password generation from the user’s own device when generating the correct 32-byte digest. Additionally, changing the random numbers that are part of the message into the cryptographic hash function can result in a completely different password from the same graphical password input. Along with the RNG values, more information could be added to the hash function to create a different password with the same graphical password input. Specifically, input to the cryptographic hash function in the device could also include the URL of the Web site login page.

If the user wants to change their password, the user would need to notify the client system which subsequently would tell the USB device to change the RNG values it currently uses as the input to the cryptographic hash function. The last RNG value for the password would be stored in a special place in flash memory and used each time the password is required. This way, the user would not need to learn a new set of click-points for a new text password or AES key. Since an important property of the cryptographic hash function is to change the output dramatically with as little as a one-bit change, any new RNG values will generate a completely new 32-byte message digest for either an AES key or text password. This would also be the mechanism used by the client system to send encrypted messages to the server using many different AES keys. The device will keep changing the RNG values for each request and a new AES key is the result. If an AES key and a text password are generated independently, RNG values must be stored separately as well. It is important to note here that the RNG values used are created at the factory by a true random generator because the device is only capable of pseudo random (PRNG) numbers. A PRNG value, however, can be used to index into a series of factory installed random numbers and to increment from a base RNG number. That is, an RNG value added to a PRNG value can result in a larger set of numbers than what can be preinstalled (Figure 20).

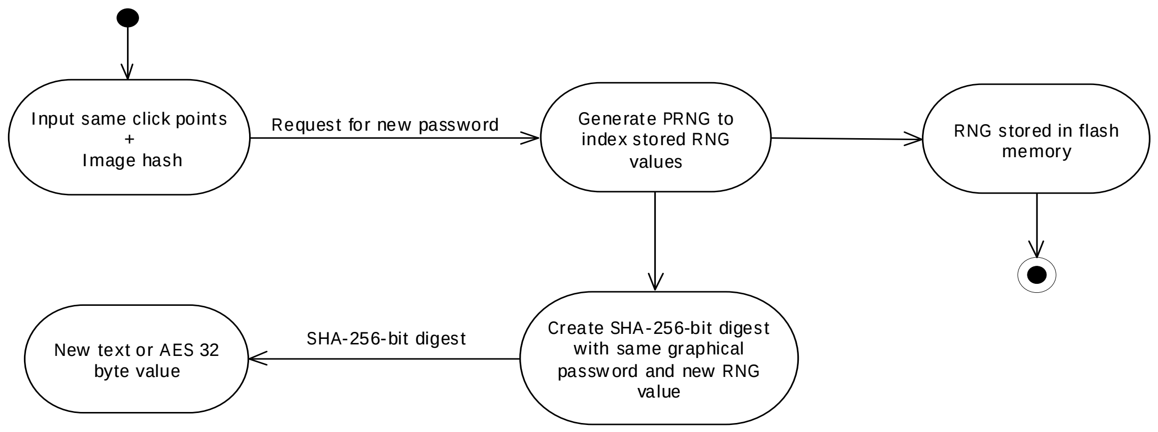

Input same click points
+
Image hash
Request for new password
Generate PRNG to index stored RNG values
RNG stored in flash memory
Create SHA-256-bit digest with same graphical password and new RNG value
SHA-256-bit digest
New text or AES 32 byte value

Figure 20: The same graphical password using different RNG values to form a new password.

Changing the password is necessary for better security but this does not take into consideration the need for multiple passwords for multiple logins. One password used for all Web sites is a potential weakness should it become compromised. A better solution would be to have a different password for every Web site. The client system could help the user keep a different password for each Web login using the root URL as another input to the cryptographic hash function. Once the user chooses a Web site to login, the password click-points and URL are sent to the device. By definition, a URL is unique and would result in a different message digest. Now, the URL is added to the input with RNG values as before. A different 32-byte message digest will result for each URL and hence, a different text password can be generated without the user changing the graphical password input (Figure 21). The device can also store the URL so the user will not have to type it the next time. Storing the URL might preclude mistyping the URL and landing on a phishing site. The application could also have a heuristic mechanism to scan suspicious URLs and warn the user. However, any verification of the URL is vulnerable to DNS poisoning and ARP spoofing.

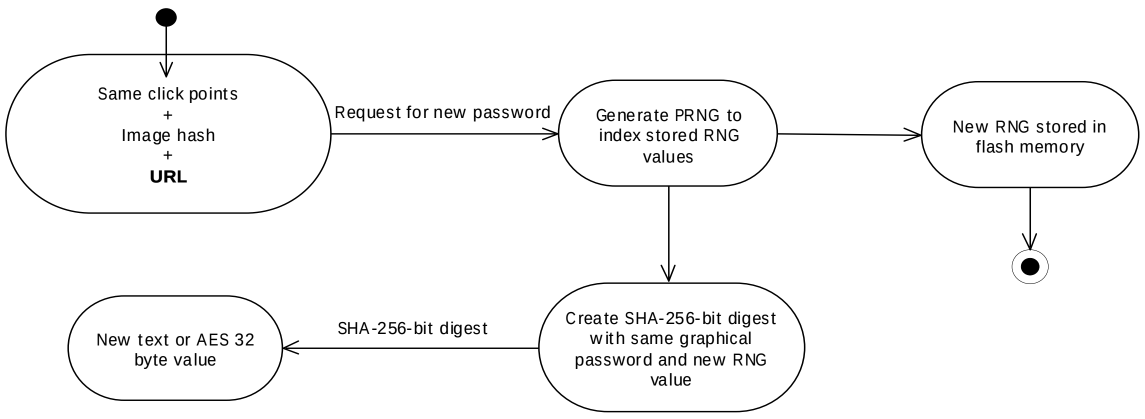

Same click points
+
Image hash
+
URL
Request for new password
Generate PRNG to index stored RNG values
New RNG stored in flash memory
Create SHA-256-bit digest with same graphical password and new RNG value
SHA-256-bit digest
New text or AES 32 byte value

Figure 21: Building a User's text password dependant on the root URL.

4.4.4 Client-Server Encryption Protocol

The public key infrastructure (PKI) cryptography used on the Internet is not as secure as once thought. Even though RSA encryption is very resistant to cryptanalysis, the transfer and certification of trusted public keys is vulnerable to man-in-the-middle (MITM) attacks. The proposed solution will use existing PKI with a second layer of cryptography to further resist MITM attacks. For this reason, AES-256 Rijndael will be used for the encryption method with constantly changing symmetric keys. Because a new AES key is sent from the client to the server at the start of each transfer, an approved key wrapping and data packaging method might be useful. The Cryptographic Message Syntax (CMS) standardizes the packaging of encrypted messages and keys with the message [86-87]. Likewise, the server should encrypt data sent to the client with the client's AES key stored in its database or the one sent with the last message. The AES key stored in the server's database for the next session should be the last AES key sent by the client (Figure 22).

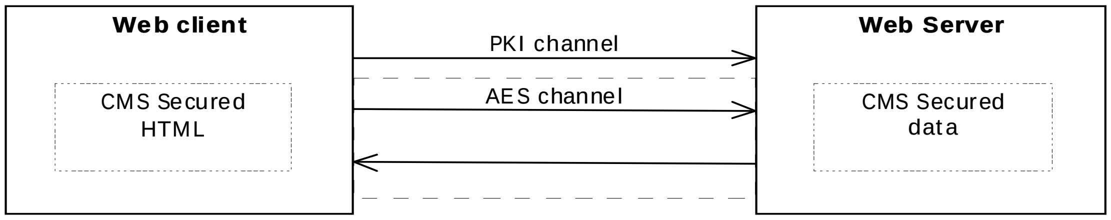


Figure 22: Overview of out-of-band client-server communication.

The proposed solution uses PKI to open an out-of-band encryption channel using AES. HTML can be sent on this channel and displayed in a separate area of the Web browser determined by the container. Since the container has access to the current Web page's HTML entities, a seamless integration of the PKI displayed HTML and the more secured AES HTML can be realized. A lightweight CMS is proposed to describe the AES encrypted data transferred in the encoded part of an HTML form. The encrypted-data content type is proposed since it handles any type of encrypted data and encryption keys to any number of recipients [86]. ASN.1 notation is used to define the structure:

```
id-encryptedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }

 EncryptedData ::= SEQUENCE {
     version CMSVersion,
     encryptedContentInfo EncryptedContentInfo
 }
```

The encrypted-data is constructed by modifying the steps in RFC 2630:

1. The encryption key will be derived from the graphical password and will be a 256-bit AES key formed using the Rijndael algorithm.
2. Key wrapping will be performed as defined in RFC 3394 [87].

3. A new AES key derived from a graphical password and SHA-256 in the USB device will be sent before the data transmission to the server encrypted with the previous AES key.
4. The server will use the last key to decrypt the new key from the client and use the client's new key to decrypt the incoming message.
5. An additional 128 bytes of padding will be added to all plain-text, to a multiple of the AES block size, and filled with random values. This includes the key wrapping.
6. The enveloped-data type will contain the session id as the recipient info.

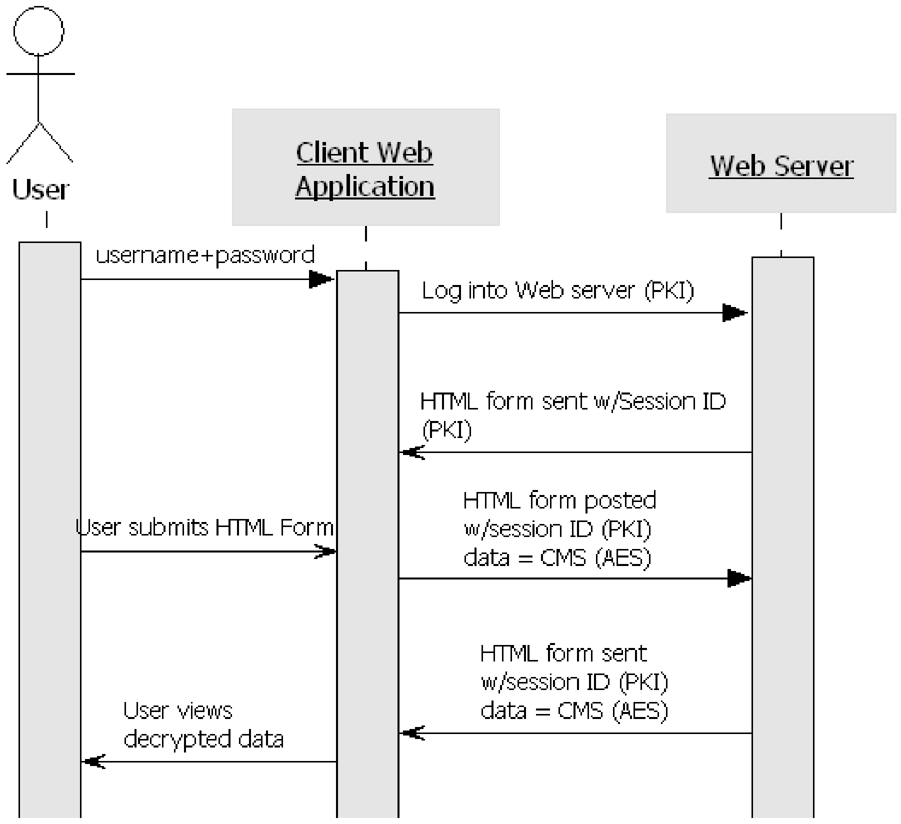

User
Client Web Application
Web Server
username+password
Log into Web server (PKI)
HTML form sent w/Session ID (PKI)
User submits HTML Form
HTML form posted w/session ID (PKI) data = CMS (AES)
HTML form sent w/session ID (PKI) data = CMS (AES)
User views decrypted data

Figure 23: Sequence of out-band-encryption.

4.5 Communicating With the Device

The Device Manager relies on the operating system's underlying support of SCSI devices. The two endpoints for all communications are the Device Manager in the client application and Crypto Device embedded in the microcontroller of the USB device. Commands/responses are sent/read in 512 bytes of encrypted data to an input/output file defined in the File Allocation Table (FAT) of the device's flash memory (Figure 24). That is, for the endpoints to communicate, one file is registered in the FAT as 512 bytes in length. There is, however, no real file. Additionally, the FAT table is formatted for 512-byte sectors of data storage. Therefore, data sent to and from the device fit into one FAT sector. In this implementation, the operating system must support the FAT specified by Microsoft [80, 81].

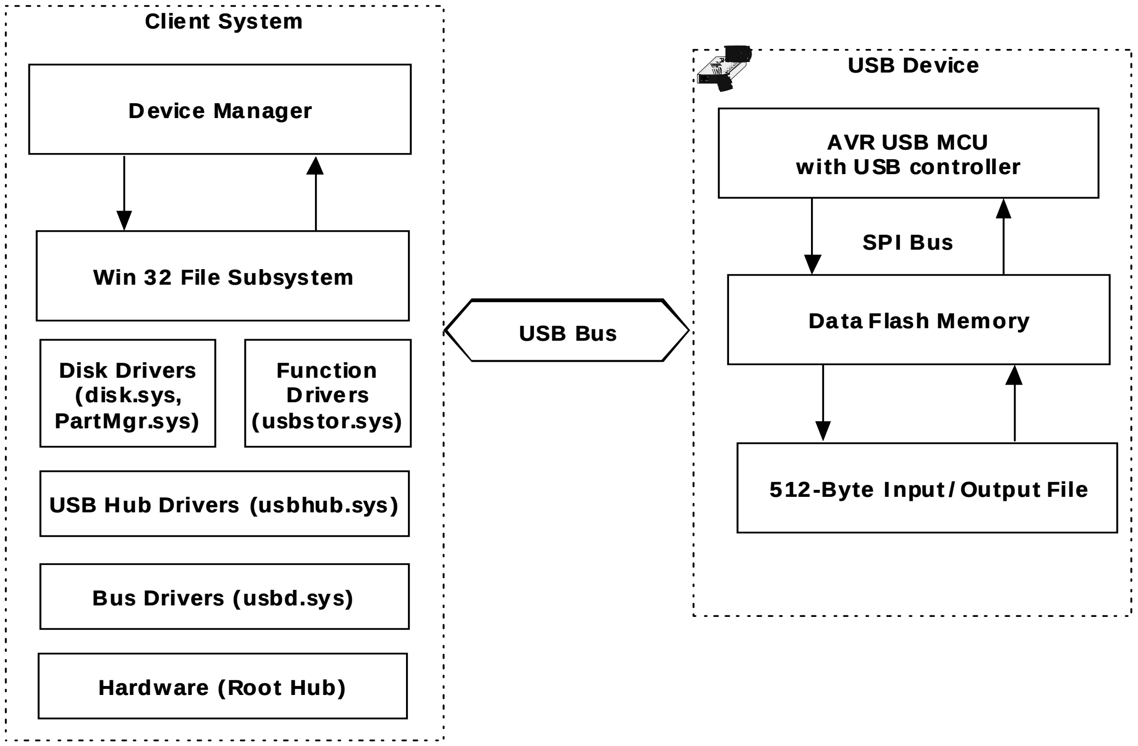

Client System
Device Manager
Win 32 File Subsystem
Disk Drivers (disk.sys, PartMgr.sys)
Function Drivers (usbstor.sys)
USB Hub Drivers (usbhub.sys)
Bus Drivers (usbd.sys)
Hardware (Root Hub)
USB Bus
USB Device
AVR USB MCU with USB controller
SPI Bus
Data Flash Memory
512-Byte Input/Output File

Figure 24: Client system to device communication block diagram [84].

The microprocessor intercepts all SCSI commands. If the client application saves data to a file on the USB device, the SCSI command will be intercepted along with the sectors of file data. Then, the microprocessor will decide from the sector address what to do with the data. If the data is in a sector that is allowed, the microprocessor will decrypt and read the data sent. If the sector is not valid or allowed, the microprocessor will throw the data into a bit bucket (an area to throw illegal sectors of data). The microprocessor will always report success for SCSI commands that contain illegal sector addresses but the microprocessor will silently throw them away which results in a simplified design. Likewise, if the client application reads a file, the SCSI command to retrieve a sector of data on the flash drive will be intercepted. If the sector is not an allowed sector, a warning text message can be sent in what would have been the file's data (Figure 25-26).

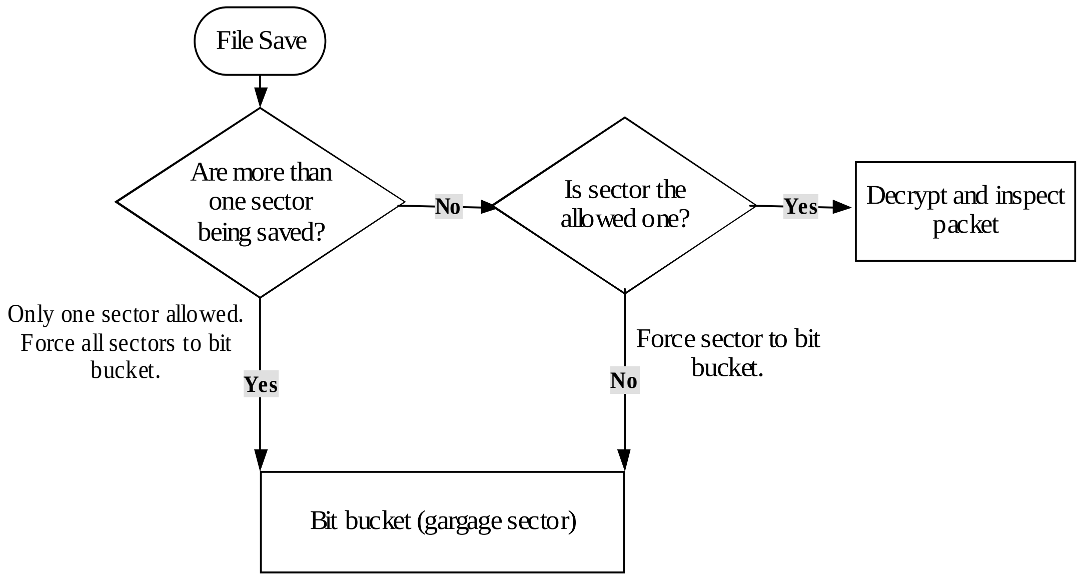

File Save
Are more than one sector being saved?
No
Is sector the allowed one?
Yes
Decrypt and inspect packet
Only one sector allowed. Force all sectors to bit bucket.
Yes
Force sector to bit bucket.
No
Bit bucket (gargage sector)

Figure 25: Overview of the action the USB device takes when the client saves data.

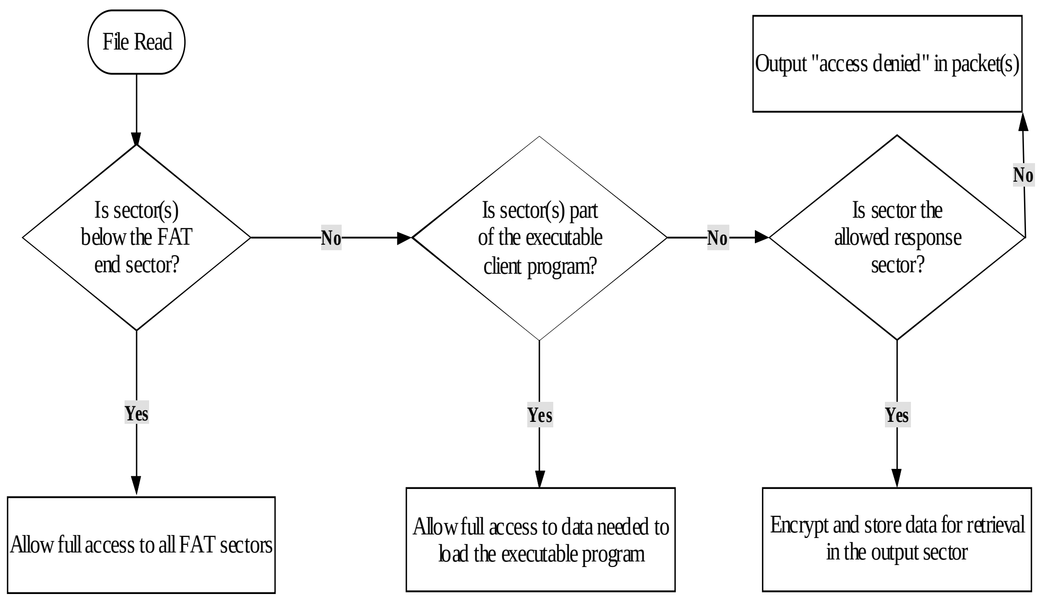
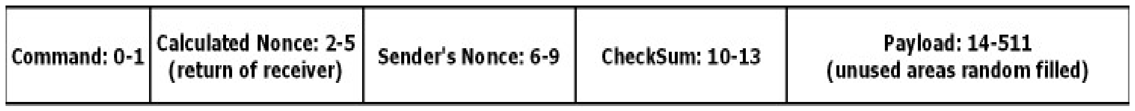

File Read
Is sector(s) below the FAT end sector?
No
Is sector(s) part of the executable client program?
No
Is sector the allowed response sector?
No
Output "access denied" in packet(s)
Yes
Yes
Yes
Allow full access to all FAT sectors
Allow full access to data needed to load the executable program
Encrypt and store data for retrieval in the output sector

Figure 26: Overview of the action the USB device takes when the client reads data.

Read-only access to the FAT table must be allowed so the input/output file and the executable can be accessed. All other sectors in the flash are protected by the microcontroller. If the input/output file is manually opened, the microcontroller will detect an invalid access, as discussed in following the sections, and return an “access denied” warning to the user and increment an internal login failure counter.

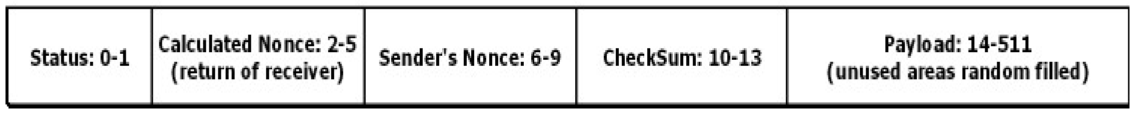
4.5.1 System Commands and Responses Summary

As was mentioned, the FAT sector is formatted to 512-byte sectors of file storage. For this reason, a 512-byte file is used for input/output. Though a file bigger than 512 bytes could have been used as the input/output, most of the data exchanged fits into 512 bytes. The exceptions are the image data and the X-Y coordinates for the graphical passwords. For those data items, multiple exchanges to the same input/output file are used. The Device Manager module of the client system sends commands to the microcontroller with the data structure in figure 27. Responses are read from the device in a data structure defined in figure 28.



Command: 0-1	Calculated Nonce: 2-5 (return of receiver)	Sender's Nonce: 6-9	CheckSum: 10-13	Payload: 14-511 (unused areas random filled)

Figure 27: Command packet sent to the device (file I/O save).



Status: 0-1	Calculated Nonce: 2-5 (return of receiver)	Sender's Nonce: 6-9	CheckSum: 10-13	Payload: 14-511 (unused areas random filled)

Figure 28: Response packet read from the device (file I/O read).

The first field indicates what action to take on the data. The next three fields are used for security and data integrity as discussed in sections 4.5.2 to 4.5.4. The last field is the payload and contains up to 498 bytes of data. Because data is streamed in a series of bytes, the byte order for the data types must also be defined. Most of the data is streamed in "Big Endian" format. However, "Little Endian" is used for the X-Y coordinates of the graphical password text. A summary of all the commands and accompanying data in the payload is outlined in 2.3 of the appendix.

4.5.2 Communication Security

Given that sensitive data is being communicated between the client application and the USB device, robust security features must be employed. Data is flowing through the operating system's file I/O system and a Trojan could inject itself as part of a system I/O module and intercept any and all file data. Hence, data from the client process to the embedded device must be encrypted. In particular, the data should be transmitted by a strong encryption algorithm, resistant to replay attacks, and resistant to implementation attacks.

AES Cipher

The encryption complies with FIPS PUB 197, AES Data Encryption Standard [73]. Data is encrypted with 256-bit AES cipher-block chaining (CBC). AES is a block cipher and the danger is that if the data is the same in each block, the output will be the same. For this reason, CBC is used so that each successive block of plain-text is XORed with the previous cipher-block [74]. However, that will not preclude a replay attack. A replay attack is where the attacker does not need to know the contents of the encrypted data to cause harm [75]. For example, if the client application sends encrypted data to the device each time the user logs in, the attacker could capture this stream of data and apply the same stream of data to the device to get the same response as the legitimate packet of encrypted data. The attacker could then log into the device merely by stealing the encrypted packet of data. For this reason, three fields, of four bytes each, have been added to the command and response data packets and discussed below (Figure 27 and 28):

- Calculated nonce.
- Sender's nonce.
- Payload checksum.

Replay Attack Prevention and PRNG Generation

To prevent replay attacks, the encrypted data can be made to have varying cipher text output for the same plain-text. To do this, random values need to be placed in or around the plain-text data. For this, nonces will are used. A nonce is a random number once used [76]. Each endpoint will create a nonce value for each plain-text payload before it is encrypted by AES. No two cipher-data transmissions between the endpoints should ever be the same. The problem is that the hardware for the USB device can only generate pseudorandom numbers (PRNG) as opposed to truly random numbers (RNG) [77]. The problem is finding a "seed" that does not fall into a finite set of values that could be guessed. Due to constraints in hardware cost, the seed value for the USB device should be stored in flash at the factory using a true RNG. Using this seed, the milliseconds from the time the device started to the time a nonce is needed can be added as seed for a better random value. The pseudocode for the PRNG of the client and device is as follows:

Client

```
srand(time(0)^CPU_ID^timerticks)
prngVal=0
while(!prngVal)
    prngVal = rand() % ((int)pow((float)2,(int)30))
```

USB Device [78]

```
RNGarray[256] = { 256 RNG values from factory }
RNGarray [timerticks%255] ^= timerticks
prngVal =RNGarray [timerticks%255]
```

Here timerticks are the milliseconds elapsed since the device was plugged into the USB port, and RNG is a true random integer value set in flash memory at the factory. The client application can use an API that generates a PRNG and can also seed this value using some hardware ID such as the CPU serial number and the time in milliseconds since the application was started.

A second layer of randomness is added to each transmission of data. With the exception of the image data that fills the whole payload, there is always some room in the payload after all the necessary items are added. In the remainder of the payload, random values from 0 to 255 are added after the valid data. With a nonce followed by random values in the remaining parts of the payload, a replay attack is extremely difficult undertake. However, the attacker could use brute force to find a break in the system. By using an automated tool and writing various streams of data to the device, it might still be possible to find an implementation weakness. Hence, there are two more fields in the data packet to further reduce implementation attacks based on brute force. One is the calculated nonce value and the other is a checksum value.

Checksum of Payload Data

The payload integrity is verified with a sensitive checksum that can find a difference even if one byte value is increased and another decreased by the same amount. A checksum should include all valid data and randomized data in the payload in order to catch any changes anywhere in the payload. If a brute force attack occurred, it would be very unlikely that the payload would checksum correctly once it was decrypted. The pseudocode for the checksum is as follows [79]:

```
len = byte length divided by 2

while len is not 0
 begin
        If len < 16,384 then
           l = len
        else
           l = 16,384
        len = len – l;
        for 0 to l
          begin
            sum = sum + data_value at index
            index = index +1
          end
      sum = (sum and HFFFF)+ (sum shift right 16)
end
 sum = (sum and HFFFF)+ (sum shift right 16)
```

Calculated Nonce Based on Shared Function and Secret

The idea of a calculated nonce is based on a shared secret that is added to part of a message to return a hard-to-guess value that will authenticate one party to another. If both system A and system B share the same function and some secret, system B sends a numeric value, X, to system A; and system A inputs the value into the shared function along with its secret and gets value Y. System B also inputs value X into its function with its secret and gets value Y. When system A responds to system B, the value Y is sent with a reply. System B then compares the calculated value from system A with its expected value. If those values match, then it is very probable that the reply from system A is authentic, provided it is hard-to-guess value Y given X. The idea can be applied to a nonce value and a common secret as an input to a shared function to get a hard to guess output value (Figure 29).

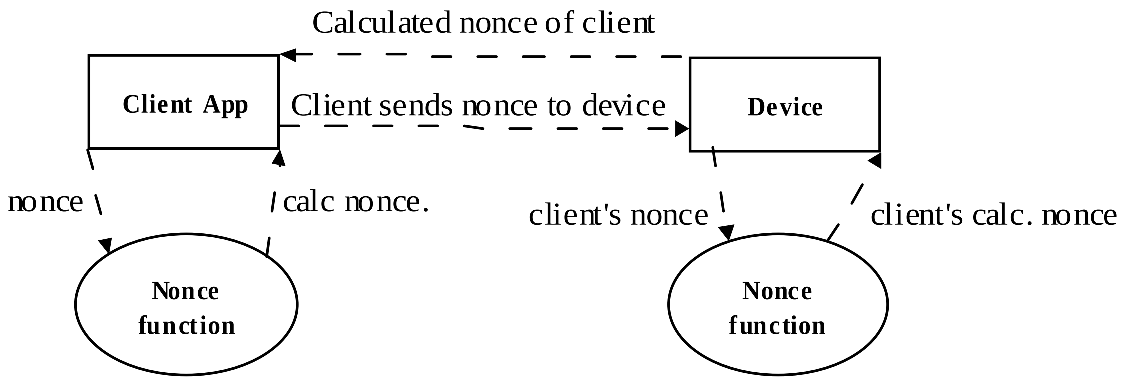


Figure 29: Nonce and calculated nonce authentication scheme.

When the client’s nonce is received by the device, the device sends the nonce into its shared function and returns the calculated nonce with the response data packet. The client application will then verify that the calculated nonce matches what it expects. Likewise, the USB device will send its nonce and store a calculated nonce value it expects to receive in the next command packet from the client. The result is that each data packet always depends on a state of the last calculated nonce they expect. These values change in every transmission, are random, and enforce a specific order of command and response. It should be reiterated that the calculated nonce is in an AES encrypted packet so it will be very hard to eavesdrop. If the calculated nonce values do not match what is expected by either party, the packet is rejected.

Client Executable and Random Checksums as the Shared Secret

The idea of a calculated nonce is a way to authenticate both parties to each other. What is required is a shared secret that, with a shared function, will generate a hard-to-guess output for authentication. For this, the client executable will be used as the shared secret and the same checksum function that checks the payload will be the shared function. Using the random nonce value discussed above, the nonce can be used as an index into the executable file to return a partial checksum (calculated nonce) defined by: $f_c(nonce, client \text{ executable}) = \text{partial checksum}$. Because the nonce value may be bigger than the executable size, a modulus of the nonce value and the file size can return a start index into the executable from which to begin a checksum. The checksum derived from a random offset into the executable to an endpoint, or end of file, will also produce a random output but not one that is difficult to guess. That is, since all of the clients run the same executable code, it might be possible to guess the calculated nonce by trying some nonce values.

The idea, then, is to not have the same executable code for each client because that would produce the same checksum. In short, the machine code in the client’s executable can be modified in random locations that still keep the operation of the executable as expected. This would produce a completely different calculated nonce value than other clients because of different executable images. This will, however, require that the microcontroller in the USB device run the same checksum on the executable file stored on its flash drive as the client will have to do on itself. For many cheaper microcontrollers, that may require too much processing speed. Therefore, it may suffice to run the checksum over one or two sectors of the stored executable so processing time is less.

Using both the calculated nonce and the payload checksum, the integrity of the payload and the authenticity of the packet can be verified. Any implementation attack would have to make sure that both the calculated nonce and the checksum are exact. If those values are not correct, the packet is rejected by the client or the device. Additionally, the device will monitor how many times it receives a bad packet of data. If the device receives too many bad packets, it can be programmed to self-destruct after so many bad packets. Consequently, any implementation attack could not use successive attacks.

Operating System Modules are Better Protected

Another defense against implementation attacks is that SCSI file I/O is a well known protocol and uses common operating system libraries. The problem with using a proprietary system is that it might have weaknesses in its implementation that have not been vetted over an extended period of time in the field. By using a well-tested protocol and libraries, accidental implementation faults are likely to be minimized. The other benefit is the exposure to anti-virus and anti-spyware applications. Since an operating system library is responsible for transferring data on the PC, a Trojan or a virus could attack it. Anti-virus and anti-spyware rely on databases that contain signatures of malware and also the correct checksums of operating system libraries. A proprietary library's checksum is unlikely to exist in a database and an anti-virus or anti-spyware program may miss the infection based purely on a checksum of the library code.

4.5.3 Device Authentication

As discussed, data packets are verified with a calculated nonce and payload checksum, however the core of the authentication revolves around the AES encryption and how the 256-bit keys are created. It should be emphasized that there is no standard login with a username and password scheme. Rather, "login" means that the device successfully decrypted the data packet and finds the expected command, payload checksum, and calculated nonce to be correct. A "login failure" is where the device decrypts the data with the expected AES key but invalid data is the result. If the device detects an invalid data packet, this is regarded as a login failure. In short, the device is constantly checking for incorrect packets and will increment the failed login count during any part of the session if it finds one (Figure 33). Above a predefined threshold of failures, the device will destroy itself.

There are two AES keys and two encryption levels when transferring data. One key is considered weak and the resultant encryption is considered unsafe but might be helpful to keep casual attackers away from the data. The other key is the strong key stored in the device and is the crucial key for encrypting/decrypting sensitive data packets. It should be noted here that the AES key stored in the device is the only piece of sensitive data and has nothing to do with the user's password for Web authentication. The AES key stored in the device is only used to encrypt data between the client application and the device.

Weak AES Key

Weak encryption is used when the client application needs the first graphical password to authenticate with the device. That is, only the graphical password image, text, and X-Y locations for the text are sent to the client application under weak encryption. In essence, if the USB device were physically stolen, the attacker would see the first graphical password screen anyway. The data during weak encryption is encrypted from a 256-bit key that is sent from inside the Portable Executable (PE) of the client application. In the PE file format, detailed by Microsoft, the first section of the file is called the DOS header and the structure is defined in 7.3 of the appendix [81].

Since Microsoft states this is a legacy section not used in Windows, the reserved words at byte offset 40 of every Windows PE can be used to store a 20-byte random value. When the microprocessor of the USB device gets the request for the first sector of the client executable file data, it can find the offset to position 40 and stuff a random, temporary AES key into the DOS header. When the client executable loads, it can check its own header, which is an offset of 40 bytes from the base address of the process, to extract the AES key to decrypt the first graphical password being sent by the device. An attacker could still capture the transferred bytes of the executable and extract the AES key. Therefore, such a scheme offers a mild form of protection against simple attacks. There is, however, a stronger security feature from this scheme.

Stuffing an AES key into the DOS header of the client executable, on the fly, creates a unique executable. No other client executable would have the same value in its header information. This AES key could also be composed of the first 20 bytes of the message digest from a SHA-256 hash, where RNG values were the message. Additionally, the AES key could expire (Figure 30). The desired result is twofold:

- The client executable will only work with the particular device it was run.
- The client executable will only work for a certain period of time before requiring a restart.

For example, if the user copied the client executable to another storage medium, from the USB device, the executable would have the temporary AES key in its header. If the user ran that executable, it would work until the key expired. Restarting the client executable outside the device would mean the device and the executable would have AES keys that are now different and hence the device can no longer communicate with the client. This will help prevent a user from using a Trojan or infected executable from outside the device.

SHA-256(RNG values): 40-59

Figure 30: AES key stored in PE header at byte offset 40 (20 bytes).

Strong AES Key

The second AES key is the one that encrypts the most sensitive information. For example, encryption with this AES key is used when the user builds their password for Web authentication. For this reason, encryption with this key is the most important to prevent attacks. Additionally, this key must be stored in the device. The user clicks on the first graphical password to build this key within the client application. If it is enforced that the user will chose a minimum of 5 text items on the graphical password, then there would be 5 text items and 5 positional values captured. That would mean 15 distinct items are unique to a password:

- 5 letters or numbers (one byte each)
- 5 x-positional values (two bytes each)
- 5 y-positional values (two bytes each)

However, the AES key used for encryption and decryption is 256 bits, 32 bytes, long. If a minimum of 15 unique values are captured, 10 of which are 2 bytes long (short), then that provides 25 bytes to fill the AES key. The remaining 7 bytes can be derived by password stretching using 15 unique values the user selected and the text used for the graphical password presentation. There are 180 text values presented to the user involving 10 digits and 26 upper and lowercase ASCII characters. An array of 180 bytes with random text stored in each byte is used to build the graphical password presentation. Therefore, an index from 0 to 179 can be used to reference the random text values. The index to this array can be derived by using what the user selected and subtract a base value and add an offset to get the index (Table 18).

Table 18: Fill last 7 bytes of AES key by mapping selected text to 180-byte array.

ASCII Text	Values	Subtract	Add	Index into 180 byte array
0 - 9	48 - 57	48	0	0 - 9
A - Z	65 - 90	65	10	10 - 36
a - z	97 - 122	97	36	37 - 63

As can be seen, password stretching can occur by using the text the user selected and the mapping into the array of random text. This will result in adding 5 more text items to the previous 25 leaving 2 more to go. Using the X-Y values, a similar mapping can occur. If the image resolution is 331x451 pixels, then the remainder from the X and Y positions divided by 180 to get a zero-based index. Additionally, adding an offset as decribed with the text will grab the other values in the 180-byte array (Figure 31).

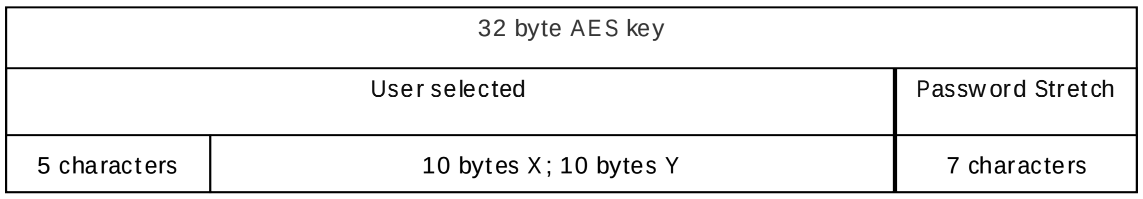


Figure 31: 256-bit AES key construction.

With two AES keys, a required sequence must take place when transferring data. As mentioned earlier, there is no distinct login command. Rather, the AES key is switched from a less secure encryption level to a higher encryption level. Once communications are switched to a higher encryption level, the level stays in effect until the client application is terminated. In Figures 32 and 33, a sequence of AES key usage is shown. Figure 33 shows what action the device will take when it receives 512 bytes of data in the allowed command file. Appendix 7.4 shows what commands are sent with what AES key.

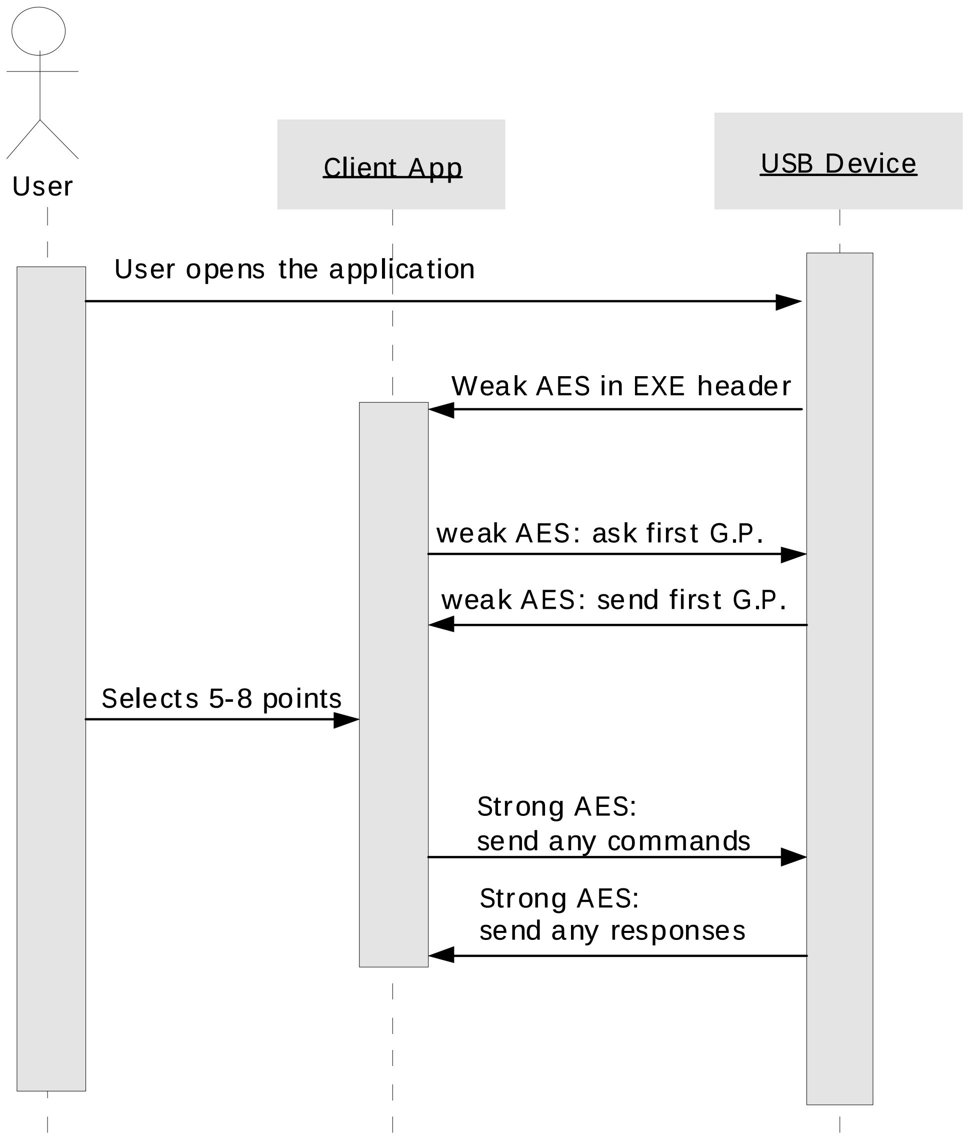


Figure 32: Sequence of AES key usage and strength.

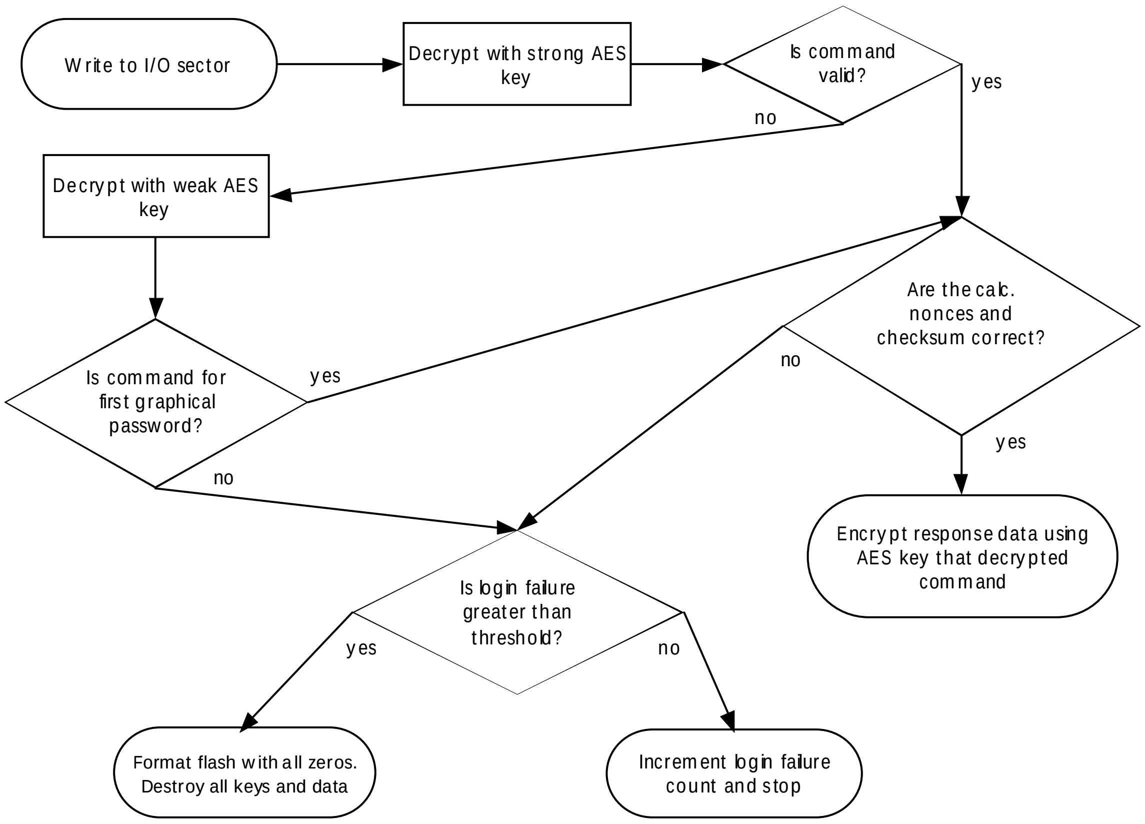

Write to I/O sector
Decrypt with strong AES key
Is command valid?
yes
no
Decrypt with weak AES key
Are the calc. nonces and checksum correct?
Is command for first graphical password?
yes
no
no
yes
Encrypt response data using AES key that decrypted command
Is login failure greater than threshold?
yes
no
Format flash with all zeros. Destroy all keys and data
Increment login failure count and stop

Figure 33: Device logic after decrypting command packet.

4.5.4 Client and Device Malware Countermeasures

The USB device is read-only, and hence, it is not possible for malware to infect the storage medium. This physical security, however, does not protect the executable before it is actually loaded into a process space and executed (Figure 34). Therefore, there are two approaches for malware countermeasures:

- Have the client executable check itself for infection.
- Have the device verify that the client executable is unaffected before it accepts any data from it.

A client checking itself for a virus is a weak form of protection if the virus has modified the executable significantly or even replaced it. Therefore, this check is only useful to warn the user. A more secure countermeasure is for the device to know if client is infected and abort or possibly self-destruct.

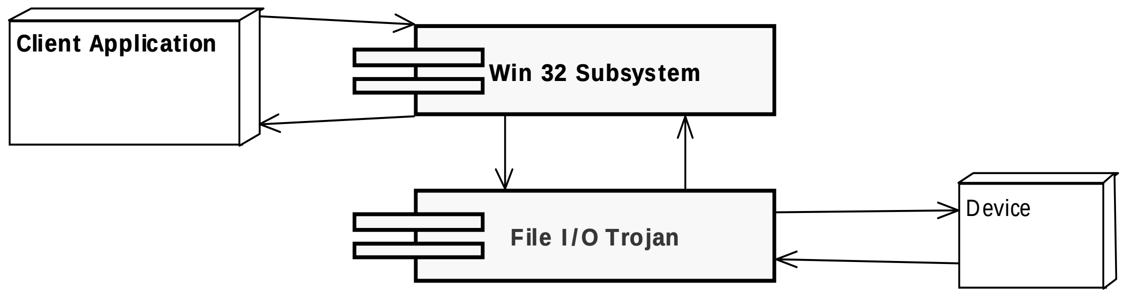


Figure 34: Trojan attached to Windows file subsystem.

Self-Checksum

It is possible for the client executable to check itself to see if the Portable Executable (PE) data has been altered [81, 82]. The checksum algorithm must be very sensitive to any change that might increment one value in the executable and equally decrement another. The client would need to compare the self-checksum with a value assumed to be correct. To find the correct checksum, the client application can query the device. Recall that the header of the PE was changed when a random 20-byte AES weak key was inserted. This addition changes the checksum of the executable. In essence, because of this numeric value, no instance of the same executable would have the same checksum. This requires that the device also run the same checksum on the executable as it leaves the device.

Client Verification using Random No Operation Instruction Insertions

A Trojan could infect the executable image as it passes through the file subsystem. To prevent this scenario, recall that calculated nonces are used to perform a packet security check based on every client executable having a unique checksum. If a virus modifies or replaces the machine code, the checksum will change and so too will the expected calculated nonce values. The desired result is that the device can detect when the client is infected and abort all communication. In order to create executables that will work exactly the same on every PC but have a unique checksum at random locations in the executable, a No Operation Instruction (NOP) machine code can be inserted into the client application source code at random locations before compilation. The NOP specifically does not change the state of any registers or data but does increase execution by 0.4-0.5 clock cycles on newer Intel processors (Figure 35) [83]. Additionally, function offsets are changed which might help obfuscate the code to an attacker looking for fixed patterns of machine code. Decoy functions could be added in many locations with unused code and a random number of NOPs. Inserting random NOPs into the source code can be done at the factory where true RNG values can be generated.

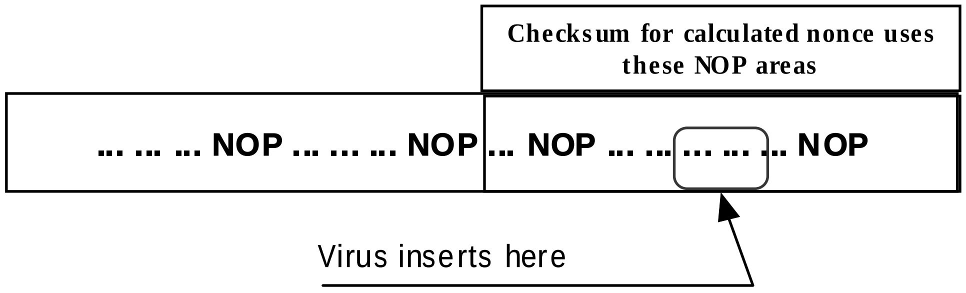

Checksum for calculated nonce uses these NOP areas
... ... NOP ... ... NOP ... NOP ... ... NOP
Virus inserts here

Figure 35: NOP opcodes in executable image at random locations.

When the device sends a nonce value to the client, a matching calculated nonce value is expected by the device. If a virus has inserted itself into the executable, the checksum will change for the calculated nonce. Because the microprocessor may not have the speed, 1024 bytes can be used for a checksum at random locations on the executable. It would be very difficult for the virus to insert without causing the calculated nonce values to be incorrect. If the device detects an unexpected calculated nonce, the device can destroy itself.

4.6 Flash Storage Map

As can be seen from the flash storage map of data, there are no passwords for Web logins stored in the device. The only semi-sensitive piece of information is the AES key used for secure communication between the device and the client application. All other data items are specific to assisting the user in creating a Web password; namely, the graphical password that the user would select their password from. The 7.5 in the appendix gives a brief description of each sector and its length. The total usage in this prototype is 775 KB out of 4 MB of flash (Figure 36).

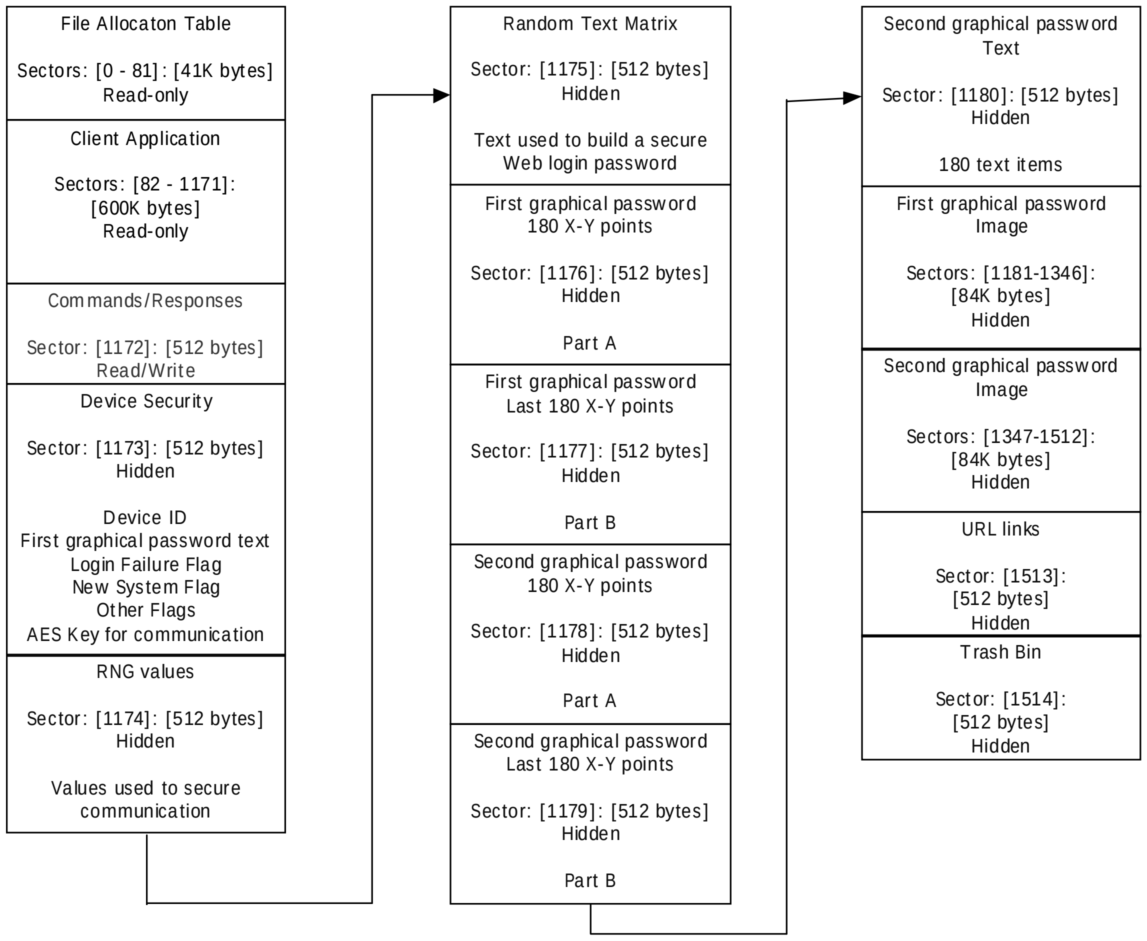


Figure 36: Flash Map of 512-byte Sectors.

CHAPTER 5 EXPERIMENTAL RESULTS

5.1 Image Hash Randomness and Uniqueness

In evaluating the image hash function, we began by assuming that any bit sequence in the resultant image hash is an independent event. Additionally, we assumed that any bit has a 50 percent chance of being a 1 or 0. Using this assumption, we should expect to see a binomial probability distribution given by the flowing equation:

$binDist(s;n,p)=\binom{n}{s}p^{s}(1-p)^{n-s}$, where s is the number of bits that are set, n is the total number of bits in the hash, and p is the probability of a bit being set. Therefore, for any given set of bits in 56 bytes (448 bits) there should be a probability distribution given by $binDist(s;448,.5)=\binom{448}{s}0.5^{448}$ if each hash value is independent from each other. Using 1,700 images, the resulting hash values were analyzed to see how the distribution of bits set compared to a binomial distribution.

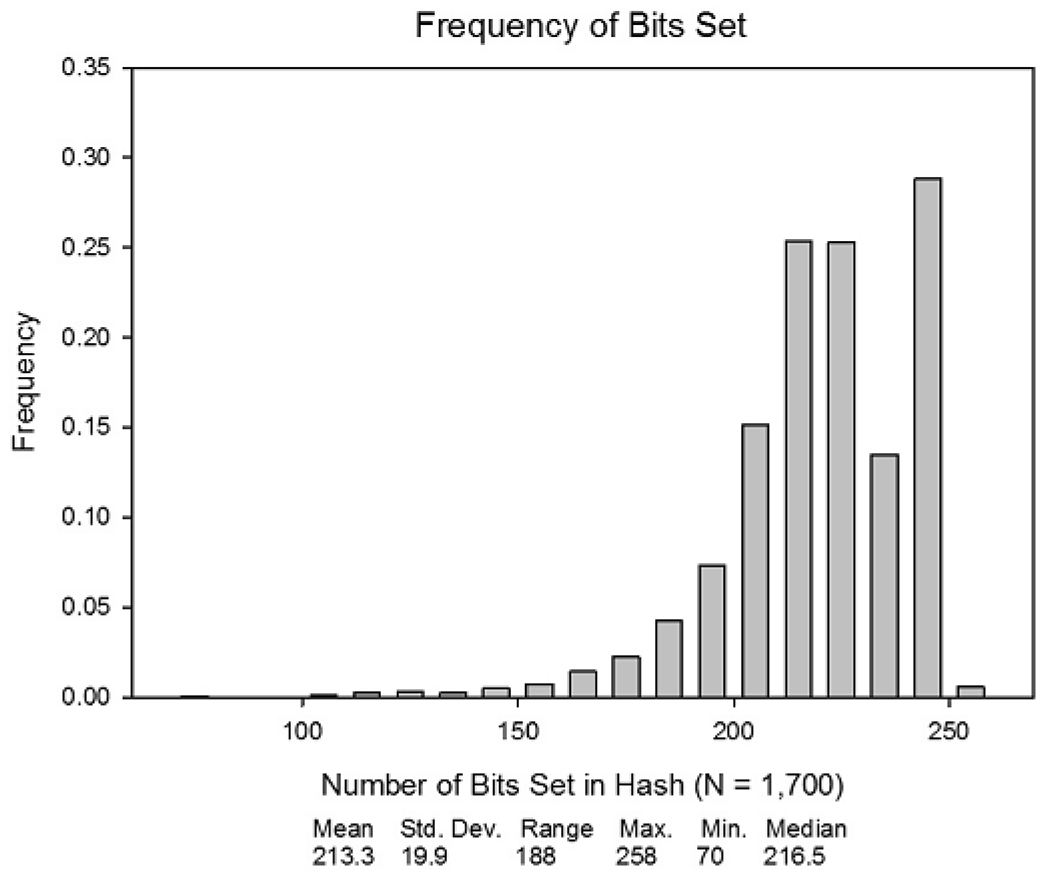
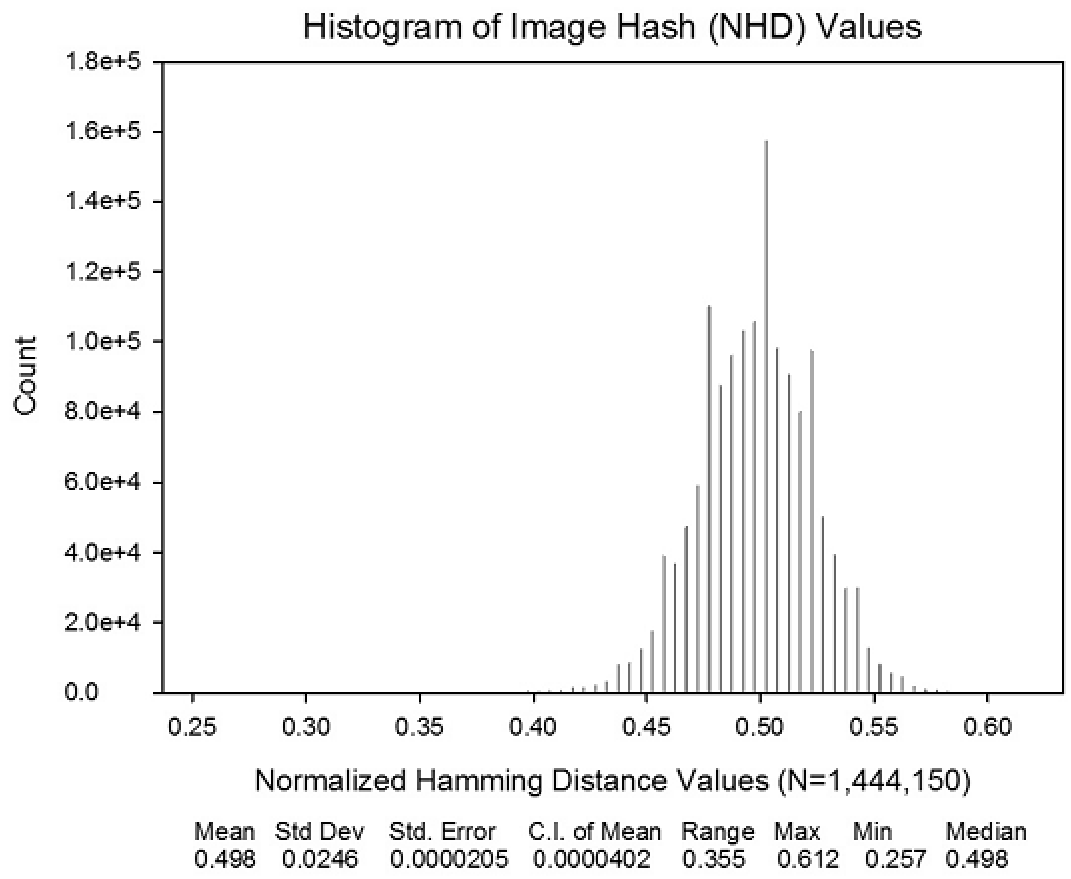


Figure 37: Distribution of bits set in image hashes from 1,700 images.

To further verify that the image hash function will produce a random, unique hash value when given unique images as an input, a relatively large set of hash-pairs will be compared. For comparison of any two image hash values, the Normalized Hamming Distance (NHD) will be used [88]:

$$NHD(H_1, H_2) = \frac{1}{L_h}\sum_{k=1}^{L_h} | H_1(k) - H_2(k) | .$$

Values of .5 or greater are expected for hash values that are different and values near zero for similar ones [88, 89]. Using 1,700 color jpeg images obtained from family photos, without any selection criteria, each image was sized to 441x331 pixels. Using the 1,700 images, associated image hash values were computed. To verify uniqueness (1,700)*(1,699)/2, or 1,444,150 unique hash-pair combinations were used to generate a histogram of NHD values.



Mean	Std Dev	Std. Error	C.I. of Mean	Range	Max	Min	Median
0.498	0.0246	0.0000205	0.0000402	0.355	0.612	0.257	0.498

Figure 38: NHD Histogram of image hash values from 1,700 images.

Once it was determined that the NHD remained near 0.5 for 1,444,150 hash-pairs, a measure of how sensitive the NHD was with a minor image change was measured. To begin, in order to eliminate any variation in measurements, all 1,700 images were saved with no intentional change using the same software that was used to make minor changes. This group of images was used as the control to compare the effect on the NHD with regard to an image change. Subsequently, a new set of images was created with a minor change. The minor change was applied equally to all pixels in the 441x331 pixel image.

Each pixel has three color channels: red, green, and blue. Each color has a value from 0 to 255 as a quantity of intensity in each channel resulting in a 24 bit value to describe a color. To determine how sensitive the NHD was to an image change, two different values were added and subtracted equally across each color channel. Using a sensitive image comparison index called the Structural Similarity Index (SSIM) described by Bovik et al., NHD changes were measured with the SSIM [90]. The values 1 and 20 were added and subtracted across each color channel and the mean and standard deviations were recorded (Table 19). Using the lowest mean value in each group, a normalized percent change was determined (Table 20).

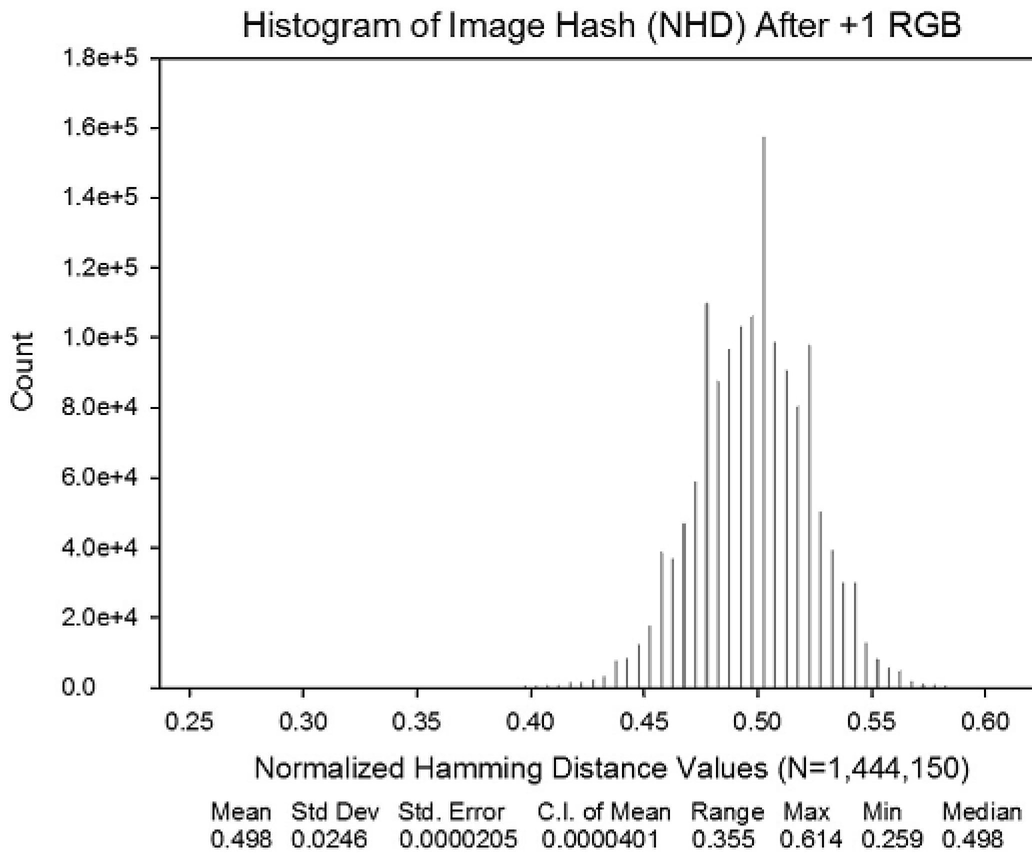
Table 19: Intensity Level versus NHD and SSIM with Control (N=1,700).

Intensity Level	**NHD μ (σ)**	**SSIM Red μ (σ)**	**SSIM Green μ (σ)**	**SSIM Blue μ (σ)**
-20	0.482 (0.0446)	0.988 (0.0103)	0.988 (0.0116)	0.987 (0.00745)
-1	0.445 (0.0460)	0.691 (0.149)	0.692 (0.141)	0.724 (0.125)
1	0.454 (0.0463)	0.981 (0.0397)	0.981 (0.0408)	0.976 (0.0411)
20	0.486 (0.0389)	0.819 (0.118)	0.818 (0.119)	0.817 (0.112)

Table 20: Normalized to Lowest Mean Value.

Intensity Level	**NHD %**	**SSIM Red %**	**SSIM Green %**	**SSIM Blue %**
-1 to -20	8.3	43.0	43.0	36.3
1 to 20	7.0	19.8	19.9	19.5

Arbitrarily choosing the image hash data created by adding one across each color channel, a histogram was created that shows the NHD of 1,444,150 hash-pairs. The result seemed similar to the NHD found from the previous histogram.



Mean	Std Dev	Std. Error	C.I. of Mean	Range	Max	Min	Median
0.498	0.0246	0.0000205	0.0000401	0.355	0.614	0.259	0.498

Figure 39: NHD Histogram of image hash values from 1,700 +1 RGB.

As seen in Figure 40, a comparison of the NHD to the control images (the ones with no change but saved with the same software) showed a slight shift below 0.5 with a mean at 0.454. There were no identical hash values in the sample set.

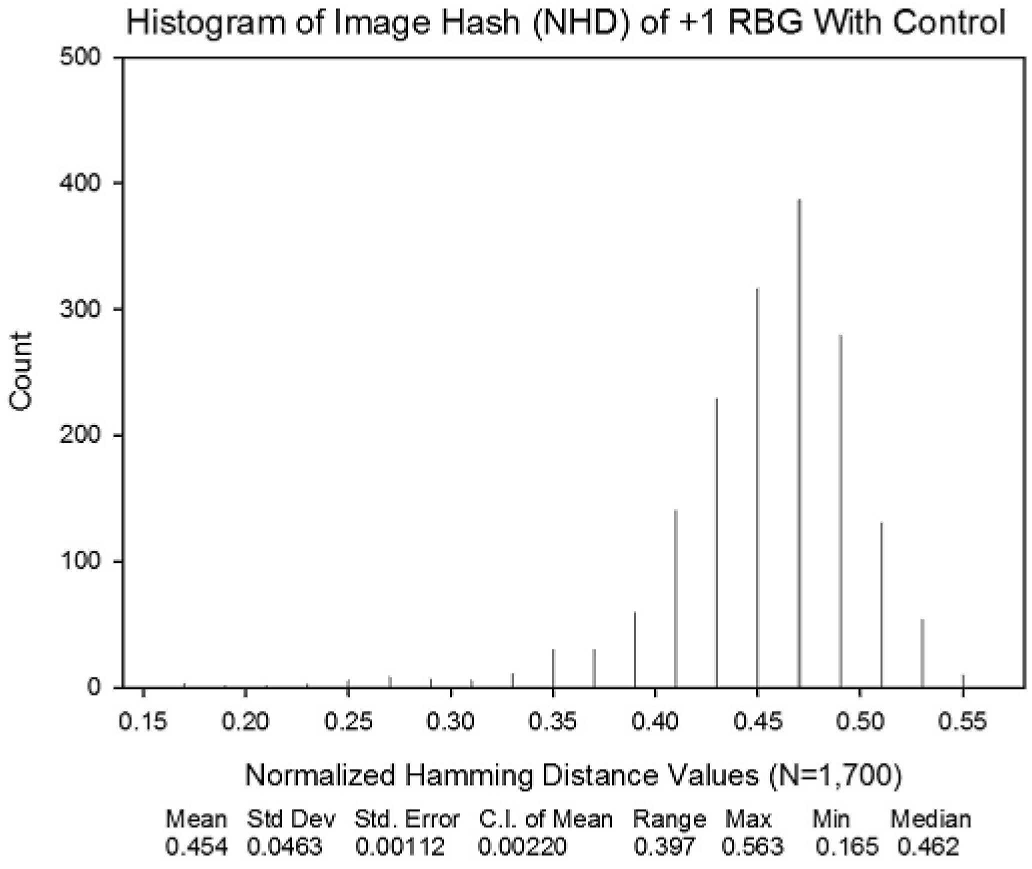
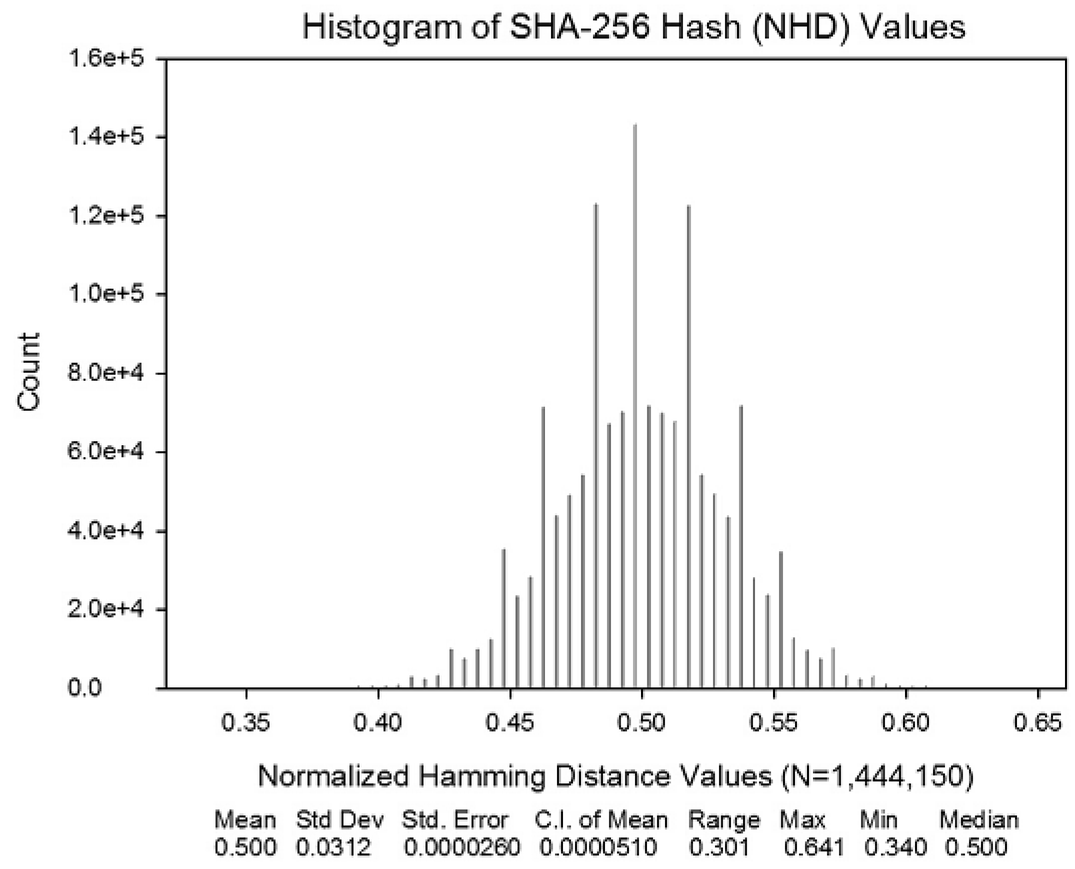


Figure 40: NHD Histogram of +1 RGB against 0 RGB.

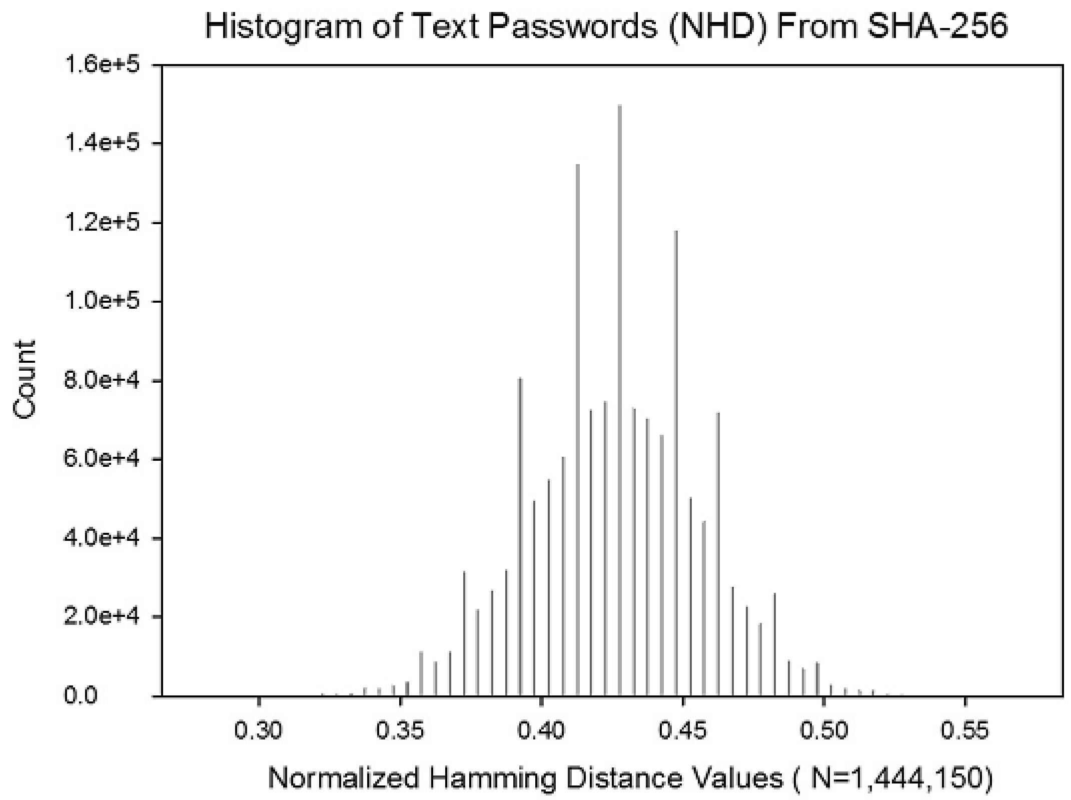
The next phase of analysis was conducted to see how unique the output from the 256-SHA function was given the image hash as an input. The 256-SHA function was expected to have a mean NHD of 0.5 and it was verified before further analysis. The NHD from the 32-byte (256 bit) message digest was measured using the same 1,700 images as the initial image hash NHD histogram. There were 1,444,150 combinations of the 32-byte hash-pairs graphed as a histogram. As expected, the 256-SHA values were centered on 0.5 (Figure 41).



Mean	Std Dev	Std. Error	C.I. of Mean	Range	Max	Min	Median
0.500	0.0312	0.0000260	0.0000510	0.301	0.641	0.340	0.500

Figure 41: NHD Histogram of 256-SHA Message Digest.

At this point, the message digest from the 256-SHA function can be used as a 256-bit AES key for secure encryption. However, since the proposed solution offers to work with existing text-based password infrastructure, a text password is needed. Though ASCII and extended ASCII have values from 0 to 255, not all characters are allowed for a text password by some applications. In order to be safe, the ASCII characters from 33 to 126 inclusive, with the 4 characters missing, were used. In total, 90 text characters were used as the text for password generation. Since the SHA-256 message digest shows randomness and uniqueness, there will be no reason to randomize the ASCII text used to map to the message digest. Rather, the value in each of the 32 bytes from the message digest will be considered as the random index into a fixed, contiguous ASCII sequence from 33 to 126, inclusive. The text value was mapped to each byte of the message digest by the modulus of the value and 90 to get the index into an array of allowed ASCII characters. Text passwords were computed from 1,700 images using the same 32-byte message digest used previously. 1,700 text passwords where used to compute a NHD as before. Each password was compared in 1,444,150 text password pairs using the NHD for 32 bytes per password. The histogram showed a mean of 0.426 and no duplicates. As expected, decreasing the allowable sequence value in each byte from 256 to 90 was a loss of entropy. However, it appeared the NHD never got below 0.281. A zero value would mean there were some passwords that were duplicates and a mean or median value much lower than 0.5 could mean many duplicates might occur in a sample population.



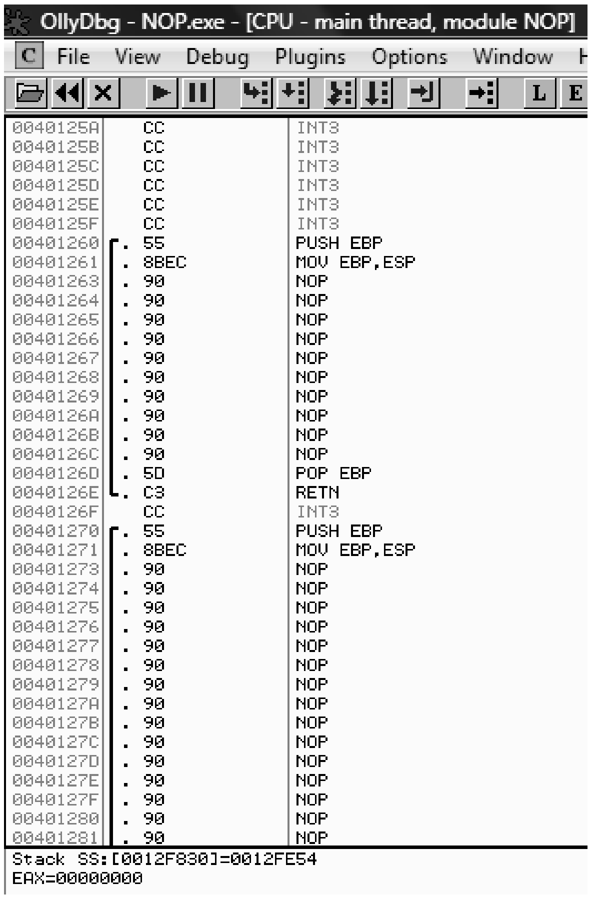
Mean	Std Dev	Std. Error	C.I. of Mean	Range	Max	Min	Median
0.426	0.0299	0.0000249	0.0000488	0.285	0.566	0.281	0.426

Figure 42: NHD Histogram of 32-Character Password.

5.2 Calculated Nonce Uniqueness and Antiviral Efficacy

In order to create a calculated nonce, the USB device will use the read-only exe file stored in its flash to compute a checksum it will expect from the client. The device will compute the checksum starting from a random offset into the stored executable file up to two sectors (1024 bytes) at a time to reduce the computational overhead. The device will use the random nonce (it will send to the client) as an index into the stored executable where the checksum should begin. The client will execute the same checksum algorithm and return the calculated nonce (checksum) to the device for verification before any command will be recognized. This scheme depends on a unique client executable per device. Therefore, if a virus attacks the client executable as it passes through the file subsystem, the checksum will change and so too will the calculated nonces that are expected by the device.

To begin the evaluation, a simple dialog-based client program was created with five empty functions in the source file. In each function of the source file, there was a tag to indicate a point for inserting random NOP assembly instructions by an automated insertion process. However, we needed a real RNG not a PRNG to generate a random number for NOP insertion. For the RNG, we used our 1,700 image hash values generated from the previous evaluation. We needed the five functions to have a random number of NOPs. Therefore, we arbitrarily chose index 0 to 4 of the image hashes for our RNG values. Values were expected to range from 0 to 255 in each of the 5 elements of the image hash. Using these values for how many NOPs to insert, insertion took place in the source file before compilation by another automated process.


OllyDbg - NOP.exe - [CPU - main thread, module NOP]
C File View Debug Plugins Options Window H
L E

```
0040125A     CC           INT3
0040125B     CC           INT3
0040125C     CC           INT3
0040125D     CC           INT3
0040125E     CC           INT3
0040125F     CC           INT3
00401260  ┌. 55           PUSH EBP
00401261  │. 8BEC         MOV EBP,ESP
00401263  │. 90           NOP
00401264  │. 90           NOP
00401265  │. 90           NOP
00401266  │. 90           NOP
00401267  │. 90           NOP
00401268  │. 90           NOP
00401269  │. 90           NOP
0040126A  │. 90           NOP
0040126B  │. 90           NOP
0040126C  │. 90           NOP
0040126D  │. 5D           POP EBP
0040126E  └. C3           RETN
0040126F     CC           INT3
00401270  ┌. 55           PUSH EBP
00401271  │. 8BEC         MOV EBP,ESP
00401273  │. 90           NOP
00401274  │. 90           NOP
00401275  │. 90           NOP
00401276  │. 90           NOP
00401277  │. 90           NOP
00401278  │. 90           NOP
00401279  │. 90           NOP
0040127A  │. 90           NOP
0040127B  │. 90           NOP
0040127C  │. 90           NOP
0040127D  │. 90           NOP
0040127E  │. 90           NOP
0040127F  │. 90           NOP
00401280  │. 90           NOP
00401281  │. 90           NOP
Stack SS:[0012F830]=0012FE54
EAX=00000000
```

Figure 43: Sample function stack with Random NOP Instructions.

Using a shareware version of OllyDbg version 1.10, it is clear that the NOPs were causing the function addresses to be in random locations. Additionally, the file size remained close or exactly the same as the original program (Figure 43). We computed the checksum on the complete portable executable (PE) image and compared the NHD with the previous evaluation. There were a couple of checksum values that were equal within the 1,700 samples; however, the mean and median showed that most NHD values were well above zero (Figure 44). The next part of the evaluation was to see if a checksum of a 1024-byte section was unique by using the NHD as a measure. For this, the first 1024 and the last 1024 bytes of the PE file were examined as with the complete checksum.

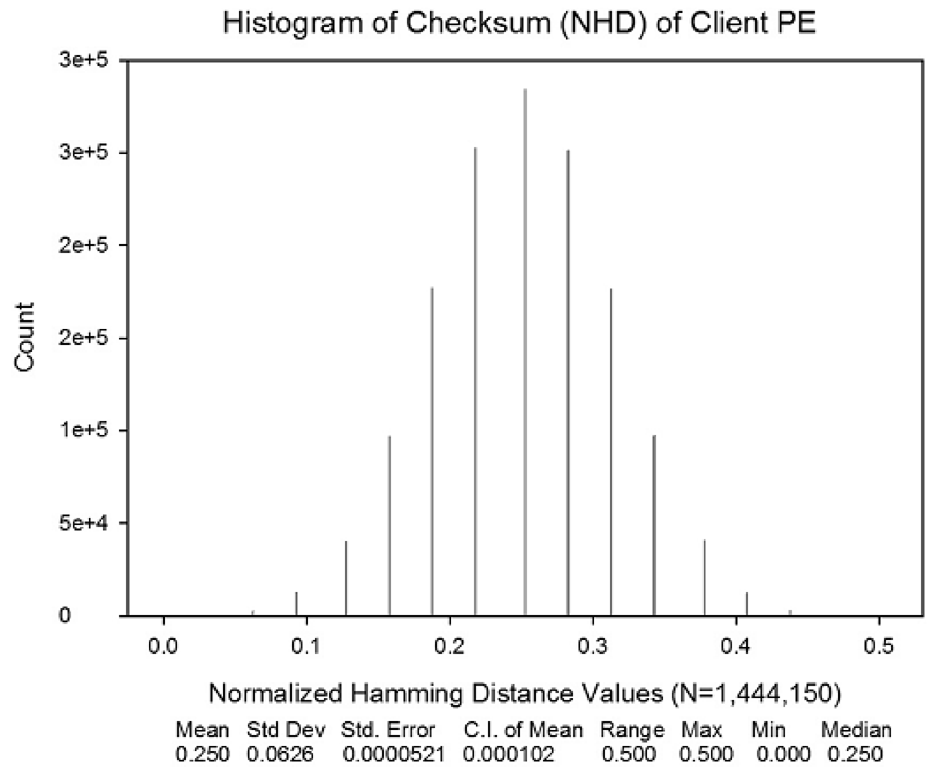


Figure 44: NHD histogram of whole PE checksum values.

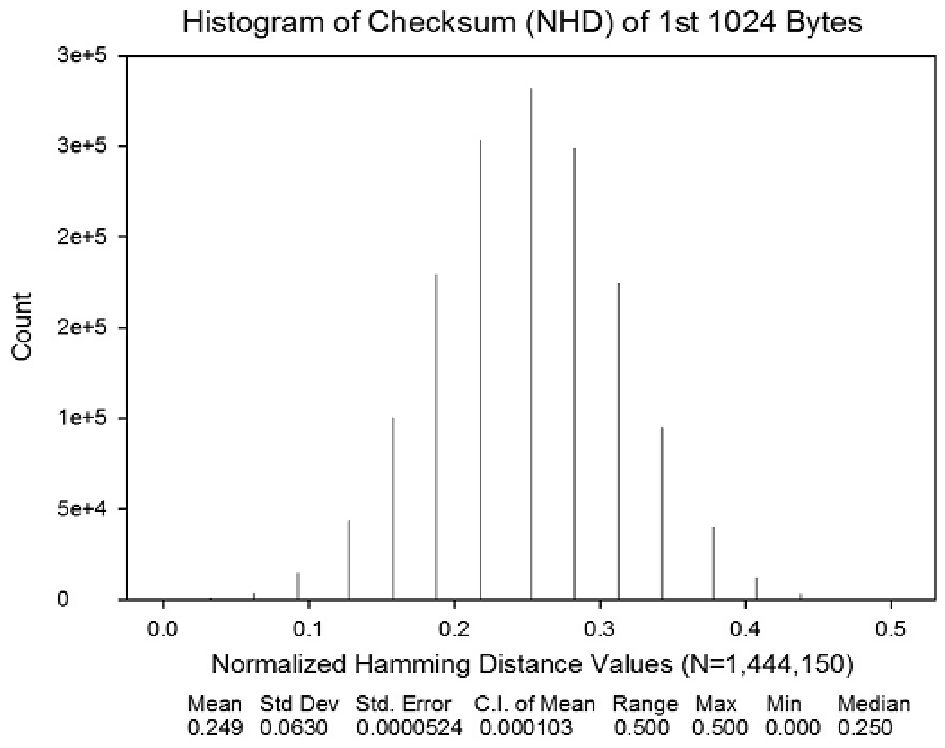


Figure 45: NHD histogram of PE checksum values for the first 1024 bytes.

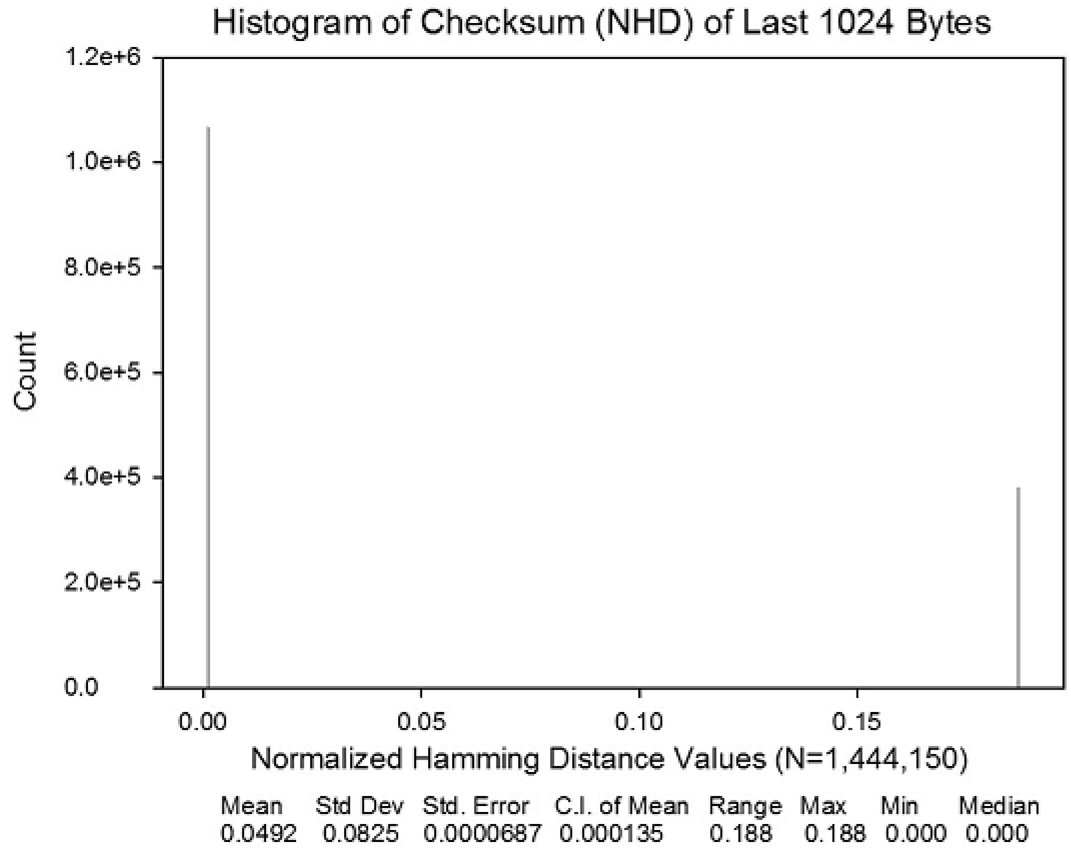


Figure 46: NHD histogram of PE checksum values for the last 1024 bytes.

The reason that the histogram in Figure 46 shows virtually all-zero NHD values is that at the end of these PE's, almost all had the same repeating values in the last 1024 bytes. However, what would happen if a virus attacked the executable and inserted itself into the last part of the file, as some viruses do? For this, we needed a virus that could infect all 1,700 executables by inserting its payload at the end of the file. We picked a virus generator called Phalcon/Skism's Gý 0.70á, *The Second Generation in Virus Creation* tool from Dark Angel. We exercised all methods of caution and ethical use of the virus. Antivirus software was constantly running to contain the spread of any infections and the test computer was disconnected from any network. The virus infected all files in the current subdirectory and encrypted its payload at the end. The virus was written in assembly and compiled and linked using Turbo Assembler (TASM). After the virus infected all 1,700 files, a NHD was computed using the checksum on the last 1024 bytes of the PE images. Now, the NHD shows mostly unique checksums where the virus attacked.

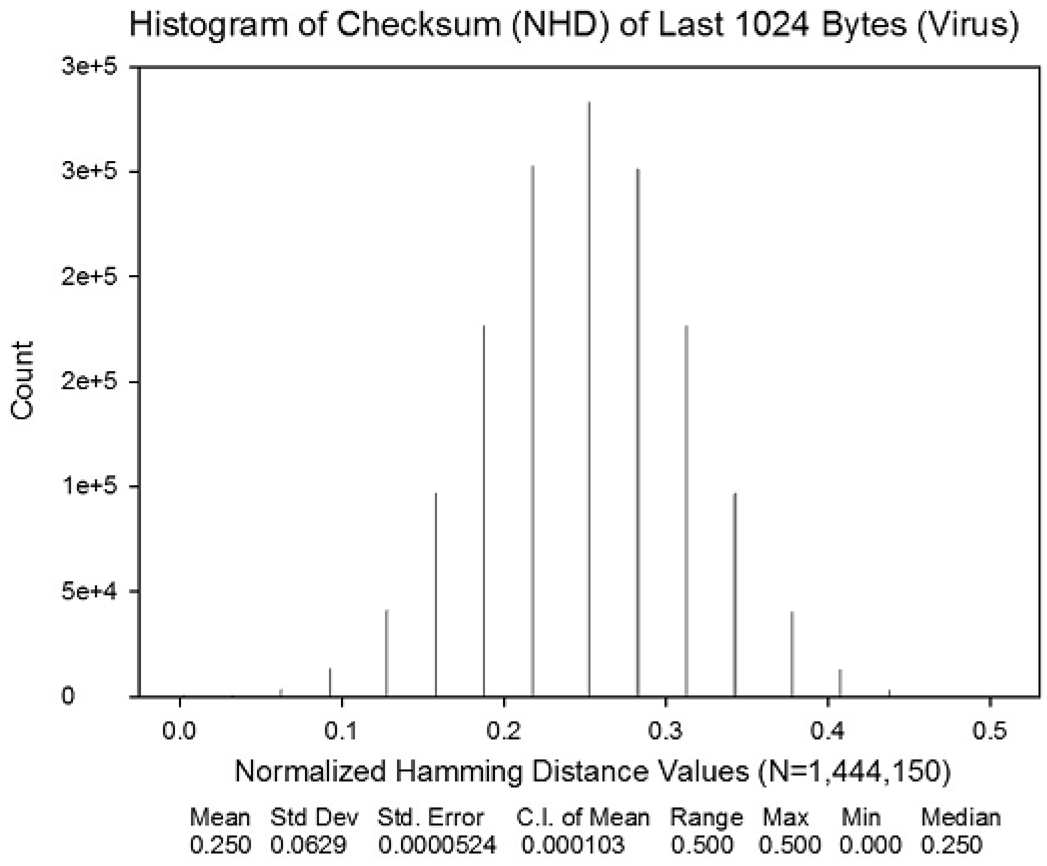


Figure 47: NHD histogram of the last 1024 bytes of the PE after a virus attack.

CHAPTER 6 DISCUSSION

In analyzing the NHD histogram from Figure 38, the proposed image hash seems to produce random and unique 56-byte hash values. The image hash also appears to be sensitive to small changes in pixel values, as demonstrated in Table 19 and 20. This implies that our image hash should be unique across a population of users if they use a personal photo as their graphical password. Further analysis verified that the SHA-256 hash, using the image hash as the message, produced a random and unique 256-bit digest (Figure 41). This should provide a strong AES encryption key. If a text password is needed, even a reduction of the allowed values from 255 to 90, for each byte in the message digest (AES key), still showed randomly distributed values and no duplicates (Figure 42).

The insertion of random NOP assembly instructions in five functions, in a test source file, did show that the function addresses are random in the executable (Figure 43). This might make it harder for viruses to find specific attack points in the client executable. More importantly, the NHD of 1,700 executables that were compiled with random NOP instructions did show that the checksums had a degree of uniqueness, though a few duplicates (Figure 44). We probably could improve the NHD results by using more functions than the five functions used in one source file. Using 1024 bytes at a time for a checksum seems to show a degree of uniqueness except at the end of the executable, which is not altered by NOP insertions (Figure 45 and 46). When the virus attacked the 1,700 executables (mostly at the end of the file), it caused unique checksums to show at the last 1024 bytes. Therefore, using checksums as calculated nonce values based on various 1024-byte locations of the client executable could help the USB device determine if the client is infected. It might also be necessary for the device to use some heuristic analysis of the client. For example, viral insertion and manipulation could increase the time the device waits to get the first command from the client.

CHAPTER 7 CONCLUSION

Research has shown that text passwords can have a large password space but very low entropy because users tend to pick passwords that can easily be guessed by dictionary style attacks. Additionally, users tend to stick with one text password that they can remember as opposed to changing their passwords regularly. Graphical passwords can also suffer from the same low entropy causes, like hot-spots, but seem to be easier to memorize. If the image of a graphical password is placed in a personal device, like a USB key or mobile device, which stays in the possession of the user, the graphical password image is not stored on the server or transmitted to the client via the network which may be more resistant to password attacks and little need for hot-spot prevention.

If the entropy of a graphical password is based only on the click-points, it could be a low entropy solution. However, if the user can pick a family photo that they keep privately in their personal device, then the entropy is very large. Using the click-points and the proposed image hash will ensure the password is unique across a very large population of users. Passing the click-points and image hash into a SHA-256 function ensures that the image details are kept hidden and a unique 256-bit value with well distributed, statistically independent bits is the result. Additionally, if the device adds random values to the SHA-256 function, the password can only be derived with the user's unique device and not another. Another advantage of adding random values, and possibly the base URL, to the SHA-256 function is that the user can change their password output while keeping the same click-points. Consequenly, the user can change their password but memorize only one graphical password.

Areas for more research are with the usability of this solution and the resistance to virus and Trojan attacks. Using NOP instructions to change the checksum may prevent some viral attacks but not all. Also, further study of the proposed solution should be measured using a quality function matrix like that proposed in A.7 of the appendix.

BIBLIOGRAPHY

[1] APWG; http://www.antiphishing.org/reports/apwg_report_Q1_2008.pdf (accessed Feb. 2009)

[2] Gartner Survey; http://www.gartner.com/it/page.jsp?id=565125 (accessed Feb. 2009)

[3] APWG; http://www.antiphishing.org/reports/apwg_report_dec_2007.pdf (accessed Feb. 2009)

[4] A. Alnajim, M. Munro; An Evaluation of Users' Tips Effectiveness for Phishing Websites Detection, Department of Computer Science*;* Durham University, ICDIM 2008, pp. 63-68

[5] T. Chen, F. Jeng, Y. Liu; Hacking tricks toward security on network environments, Department of Applied Mathematics; National Chiayi University, Taiwan, PDCAT, Dec. 2006 pp. 442-447

[6] Baig, M.M.; Mahmood, W.; A Robust Technique of Anti Key-Logging using Key-Logging Mechanism, Al-Khawarzmi Institute of Computer Science; University of Engineering & Technology, Pakistan, DEST, Feb. 2007, pp. 314-318

[7] Key Ghost; http://www.keyghost.com/specifications.htm (accessed Feb. 2009)

[8] Thawatchai Chomsiri; HTTPS Hacking Protection, Faculty of Informatics, Mahasarakham University, Thailand, AINAW, May 2007, vol. 1, pp. 590-594

[9] Sotirov, Alexander et. al., MD5 considered harmful today Creating a rogue CA certificate, Technische Universiteit Eindhoven, www.win.tue.nl/hashclash/rogue-ca , Dec. 2008 (accessed Feb. 2009)

[10] F. Callegati, W. Cerroni, M.Ramilli; Man-in-the-Middle Attack to the HTTPS Protocol; IEEE Security and Privacy; Jan. 2009;pp.78-81

[11] S. Schechter, R. Dhamija, A. Ozment,I. Fischer; The Emperor's New Security Indicators; IEEE Symposium on Security and Privacy; May 20-23 2007; pp.51-65

[12] Ors, S.B., Gurkaynak, F., Oswald, E., Preneel, B.; Power-Analysis Attack on an ASIC AES implementation, Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, Belgium, ITCC, 2004, vol 2, pp.546-552

[13] De Mulder, E. et. al., Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem, Member IEEE, EUROCON, Nov. 2005, vol. 2, pp. 1879-1882

[14] Wollinger, Paar, How Secure Are FPGAs in Cryptographic Applications?, Horst Görtz Institute for IT Security; Ruhr-Universität Bochum, Germany, 2003, http://eprint.iacr.org (accessed Feb 2009)

[15] M. Sasse, A. Adams, Users Are Not The Enemy, Communications of the ACM, Dec. 1999, vol. 42, #12, pp. 40-46

[16] Bank of America SiteKey, http://www.bankofamerica.com/privacy/sitekey (accessed Feb. 2009)

[17] Vedetta, Self-signed SSL certificates vs commercial SSL certificates: How Mozilla is killing self-signed certificates, http://www.vedetta.com/self-signed-ssl-certificates-vs-commercial-ssl-certificates-how-mozilla-is-killing-self-signed-certificates (accessed Feb. 2009)

[18] Mozilla Firefox help, http://support.mozilla.com/en-US/kb/Options+window#Security_Warnings_Dialog (accessed Feb. 2009)

[19] G. DeLone, L. Hughes; Viruses, Worms, and Trojan Horses: Serious Crimes, Nuisance, or Both? University of Nebraska at Omaha; Social Science Computer Review; 2007, vol. 25,#1,pp. 78-98

[20] P. Ször,P. Ferrie; Hunting For Metamorphic, Symantec, white paper http://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf (accessed Feb. 2009)

[21] Stanford University, CS 201, How Anti-Virus Software Works, http://www-cse.stanford.edu/classes/cs201/projects-00-01/viruses/anti-virus.html (accessed Feb. 2009)

[22] Anti-malware Test Lab, http://www.anti-malware-test.com/?q=node/39 (accessed Feb. 2009)

[23] Av-comparatives.org, http://www.av-comparatives.org/seiten/ergebnisse_2008_11.php (accessed Feb. 2009)

[24] A. Forget, R. Biddle; Memorability of Persuasive Passwords, Carleton University, CHI, Apr 5-10, 2008

[25] Jianxin Jeff Yan; A note on proactive password checking, Proceedings of the 2001 workshop on New security paradigms, Cloudcroft, New Mexico, Session 7, pp.127-135

[26] J. Yan,,A. Blackwell,R. Anderson,A. Grant; The memorability and security of passwords - some empirical results, University of Cambridge, Sept.2000, Technical Report #500

[27] Kuo, Romanosky, Cranor, Human Selection of Mnemonic Phrase-based Passwords, ACM International Conference Proceeding Series, 2006, vol.149, pp. 67-78

[28] Forget, Chiasson, Van Oorschot, Biddle, Improving text passwords through persuasion, Carleton University, 2008, ACM, Proceedings of the 4th symposium on Usable privacy and security, pp. 1-12

[29] Securitas Operandi, Password Safe, http://peterhgregory.wordpress.com/2007/02/27/use-password-safe-to-manage-passwords (accessed Feb. 2009)

[30] Password Safe Pro, http://www.passwordsafepro.com (accessed Feb. 2009)

[31] Password Safe, http://passwordsafe.sourceforge.net (accessed Feb. 2009)

[32] SecLists.Org, Vulnerability Development: PasswordSafe 3.0 weak random number generator allows key recovery attack, http://seclists.org/vuln-dev/2006/Mar/0013.html (accessed Feb. 2009)

[33] D. Nelson, U. Reed, and J. Walling – Picture Superiority Effect, Journal of Experimental Psychology; Human Learning and Memory, 1977, vol. 2, No. 5, pp. 523-528

[34] Chen, Eng, Jiang, Visual working memory for trained and novel polygons, Harvard University, VISUAL COGNITION, 2006, 14(1), pp.37-54

[35] G. A. Miller, The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, The Psychological Review, 1956, vol. 63, pp. 81-97

[36] R. Dhamija, A. Perrig, Déjà vu: A User Study Using Images for Authentication, Proceedings of the 9th USENIX Security Symposium, 2000, pp. 45-58

[37] Passfaces™, www.passfaces.com (accessed Feb. 2009)

[38] S. Brostoff, M. Sasse, Are Passfaces ™ more usable than passwords? A field trial investigation, Proceedings of HCI, Sept. 2000, pp. 405-424

[39] De Angeli, et al., Is a picture really worth a thousand words? Exploring the feasibility of graphical authentication systems, University of Glasgow, International Journal of Human-Computer Studies, Jul. 2005, vol. 63, pp. 128-152

[40] Susan Wiedenbeck et al., PassPoints: design and longitudinal evaluation of a graphical password system
International Journal of Human-Computer Studies, Jul. 2005, vol.63, pp. 102-127

[41] Chiasson,Biddle,Oorschot, A Second Look at the Usability of Click-Based Graphical Passwords, ACM International Conference Proceeding Series, 2007, Vol.229, pp. 1-12

[42] Treil, Mallat, Image Wavelet Decomposition and Applications, University of Pennsylvania, Technical Reports (CIS), Department of Computer & Information Science,1989

[43] Bhattacharjee, S.; Kutter, M., Compression Tolerant Image Authentication, ICIP, Oct. 1998, vol.1, pp. 435-439

[44] Wu, Min; Mao, Yinian; Swaminathan, Ashwin; A Signal Processing and Randomization Perspective of Robust and Secure Image Hashing, SSP, Aug. 2007, pp. 166-170

[45] General Purpose Hash Function Algorithms, http://www.partow.net/programming/hashfunctions/index.html#HashingMethodologies (accessed Feb. 2009)

[46] Digital Watermarking Alliance, http://www.digitalwatermarkingalliance.org (accessed Feb. 2009)

[47] Microsoft, System and Method for Hashing Digital Images, www.wipo.int/pctdb/en/wo.jsp?wo=2002037331 (accessed Feb. 2009)

[48] P.W. Wong, "A Watermark for Image Integrity and Ownership Verification", 1997, Proc. IS&T PIC Conf

[49] Ahmed, F., Siyal, M.Y.,A, Secure and Robust Hashing Scheme for Image Authentication, ICICS.2005, pp. 705-709

[50] Patrick Elftmann, Secure Alternatives to Password-based Authentication Mechanisms, RWTH Aachen University, Diploma Thesis, Aachen, Germany, Oct. 2006

[51] Tari, Ozok, Holden, A Comparison of Perceived and Real Shoulder-surfing Risks between Alphanumeric and Graphical Passwords,ACM,2006, vol.149,pp. 56-66

[52] Komanduri,S.,Hutchings,D, ACM International Conference Proceeding Series, Bowling Green State University, 2007,vol.229, pp.20-28

[53] Chiasson, Forget, Oorschot, Influencing Users Towards Better Passwords: Persuasive Cued Click-Points, ACM, 2008, pp. 1-12

[54] Forget, Chiasson, Oorschot, Improving Text Passwords Through Persuasion, ACM, 2008, pp. 1-12

[55] Thorpe, Van Oorschot, Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords. USENIX
Security, 2007

[56] Doja, Kumar, Image Authentication Schemes against Key-Logger Spyware, ACIS, Aug. 2008, pp. 574-579

[57] Beyond If Solutions, http://www.beyondifsolutions.com (accessed Feb. 2009)

[58] UNISYS: European Biometrics Portal Biometrics in Europe Trend Report 2007, http://www.eubiometricsforum.com/pdfs/TrendReport2007.pdf (accessed Feb. 2009)

[59] Microsoft Fingerprint Reader, http://www.amazon.com/Microsoft-DG2-00002-Fingerprint-Reader/dp/B0002WPSB2 (accessed Feb. 2009)

[60] AuthenTec 2004 Consumer Biometrics Survey, http://www.authentec.com/docs/consumersurvey.pdf (accessed Feb. 2009)

[61] Biometric Survey Results, Oct. 2002, http://www.biometrie-online.net/dossiers/generalites/Results10-02%5B1%5D.pdf (accessed Feb. 2009)

[62] Lawrence O'Gorman, Comparing Passwords, Tokens, and Biometrics for User Authentication, IEEE Fellow, Proceedings of the IEEE, Dec 2003, vol.91, No.12

[63] Hardware token, http://en.wikipedia.org/wiki/Security_token

[64] Software token, http://en.wikipedia.org/wiki/Software_token

[65] Federal Financial Institutions Examination Council, http://www.ffiec.gov/pdf/authentication_guidance.pdf (accessed Feb. 2009)

[66] Easy Web login, TD Canada Trust, https://easyweb.tdcanadatrust.com (accessed Feb. 2009)

[67] Smart card readers, http://www.nextag.com/usb-smart-card-reader/search-html (accessed Feb. 2009)

[68] IronKey USB Token, http://www.ironkey.com (accessed Feb. 2009)

[69] Yubico USB Token, http://yubico.com/products/review/ (accessed Feb. 2009)

[70] SCA Analysis, http://www.iaik.tugraz.at/content/research/implementation_attacks/introduction_to_impa (accessed Feb. 2009)

[71] http://en.wikipedia.org/wiki/Cryptographic_hash_function (accessed Feb. 2009)

[72] http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf (accessed Feb. 2009)

[73] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (accessed Feb. 2009)

[74] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_.28CBC.29 (accessed Feb. 2009)

[75] http://en.wikipedia.org/wiki/Replay_attack (accessed Feb. 2009)

[76] http://en.wikipedia.org/wiki/Cryptographic_nonce (accessed Feb. 2009)

[77] http://en.wikipedia.org/wiki/Pseudorandom_number_generator (accessed Feb. 2009)

[78] http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf (accessed Feb. 2009)

[79] http://www.faqs.org/rfcs/rfc1071.html (accessed Feb. 2009)

[80] http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx (accessed Feb. 2009)

[81] http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx (accessed Feb. 2009)

[82] http://msdn.microsoft.com/en-us/library/ms679281(VS.85).aspx (accessed Feb. 2009)

[83] http://developer.intel.com/design/pentiumii/manuals/243191.htm (accessed Feb. 2009)

[84] http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3879 (accessed Feb. 2009)

[85] http://msdn.microsoft.com/en-us/library/ms644990(VS.85).aspx (accessed Feb. 2009)

[86] http://www.ietf.org/rfc/rfc2630.txt (accessed Feb. 2009)

[87] http://www.faqs.org/rfcs/rfc3394.html (accessed Feb. 2009)

[88] A.Swaminathan, Y. Mao, M.Wu; Robust and Secure Image Hashing; IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 1, NO. 2, JUNE 2006

[89] S. Xiang, H. Kim, J. Huang; Histogram-Based Image Hashing Scheme Robust Against Geometric Deformations; ACM, pp: 121-128, 2007

[90] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.

APPENDIX A

A.1 Hardware Design

An Atmel 8-bit AVR MCU with an on-chip USB controller was selected to simplify design and allow for small circuit size [84]. Atmel was chosen because of plenty of supporting documentation and firmware source code to support fast design and implementation. Additionally, Atmel provided free serial flash memory samples. The constraining requirements of the project were program and data memory size, processor speed, and cost.

A.1.1 Microcontroller

An Atmel AT90USB128 8-bit microcontroller with 128K bytes of ISP flash and on-chip USB controller running on an 8 MHz clock was used. The design had a clock frequency of 8 MHz because the data sheet specifies a clock frequency based on V_{cc}. The AT90USB128 is a RISC CPU with in-system programmable flash. The device was programmed and debugged in circuit with an IEEE 1149.1 standard JTAG interface using an AVR® JTAGICE mkII from Atmel. Firmware was small enough to fit into 64K bytes but 128K ISP flash was used for the design. Programming was accomplished via a 4-pin JTAG port using TCK, TMS, TDI, and TDO on the PF port of the MCU (Table 21). The USB controller was connected to a modified A-male USB connector with the following pin layout, wire color, and MCU connections as described in table 22. The serial peripheral interface bus was used to communicate between the MCU and the flash memory storage as defined in table 23.

Table 21: Port F (JTAG programming & debugging)

PIN	**Function**
PF7	TDI JTAG Test Data Input
PF6	TDO JTAG Test Data Output
PF5	TMS JTAG Test Mode Select
PF4	TCK JTAG Test Clock

Table 22: USB pin configuration (http://pinouts.ws/usb-pinout.html)

PIN	**Signal**	**Color**	**Description**	**MCU Pin**
1	V_{cc}	Red	+5V	UVCC,AVCC,DVCC, UVCON,VBUS, JTAG
2	D-	White	Data -	D-
3	D+	Green	Data +	D+
4	Gnd	Black	Ground	PCB ground plane

Table 23: Port B SPI connection

PIN	**Function**
PB1	SCK (SPI Bus Serial Clock or Pin Change Interrupt 1)
PB2	PDI (Programming Data Input or SPI Bus Master Output/Slave Input or Pin Change Interrupt 2)
PB3	PDO (Programming Data Output or SPI Bus Master Input/Slave Output or Pin Change Interrupt 3)
PB4	OC2A (Output Compare and PWM Output A for Timer/Counter2 or Pin Change Interrupt 4)

Using JTAG, on-chip flash can be verified, programmed, and locked. The MCU can be secured programmatically by setting the correct value in the SPMCSR register to lock or prevent the boot loader and application sections from any software changes. However, because this paper uses a test design, none of the lock bits were set and the program contents can be easily extracted.

A.1.2 Flash Memory

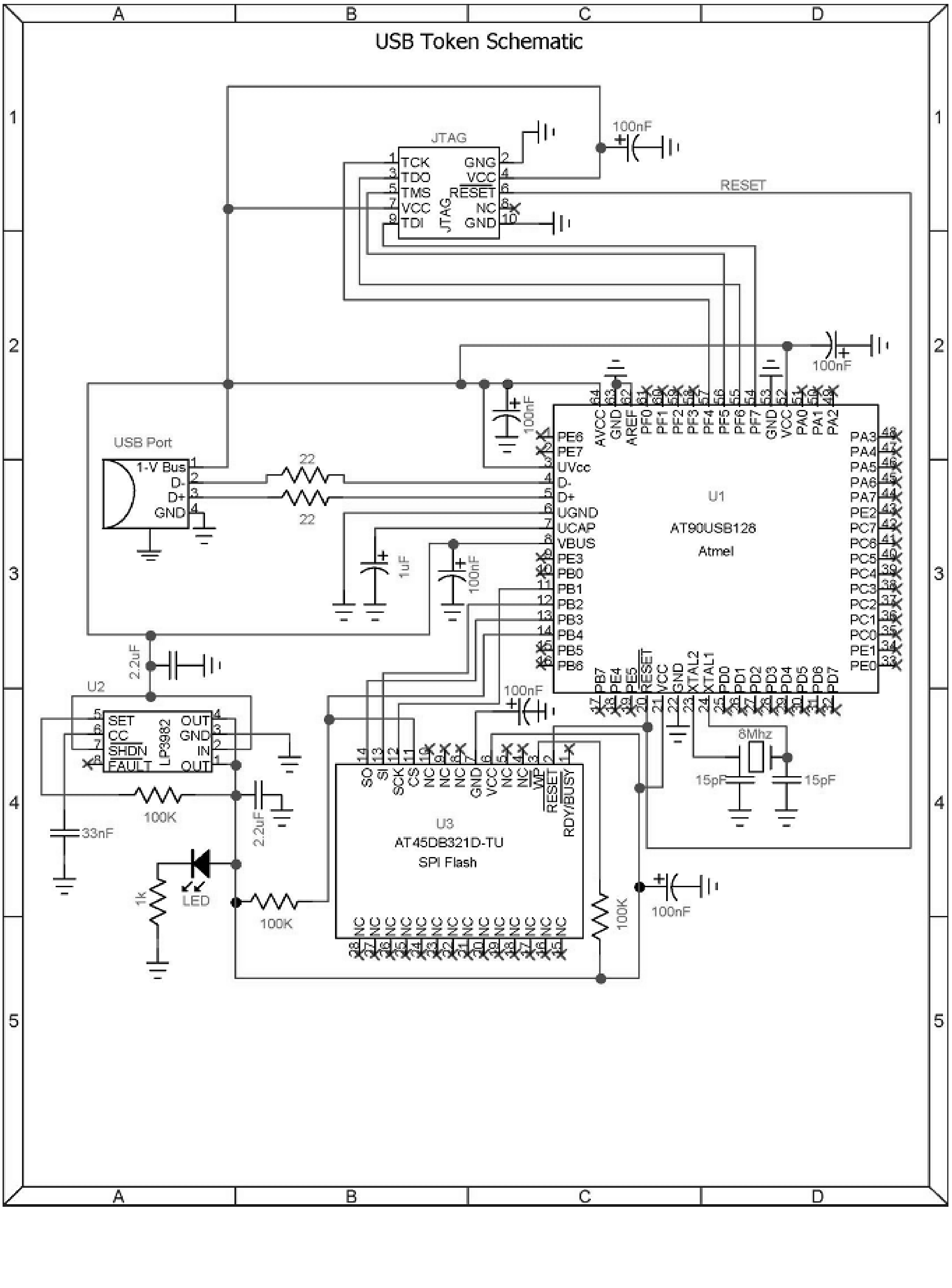
One Atmel AT45DB321C, 2.7 volt, serial flash was used as the mass storage device. The 34,603,008 bits, 4 MB, of memory are organized into 8192 memory pages of 528 bytes each. For this reason, the minimum cluster size for the FAT 32 mass storage was set to 512 bytes for better memory storage efficiency. Because pages of flash memory can hold a cluster of mass storage data, operations such as page-erase, opcode 81H, allows the flash to destroy data quickly. Quick data removal is necessary for tamper detection and data destruction. Though it is possible for data bits to be detected by cryptanalysis methods, data-remanence will be mitigated by applying some periodic bit-flipping of stored bits in data sensitive pages. It may be necessary to design a mechanical means of tamper proofing memory using a metal shield, strong epoxy, and chemical zeroization. In this solution, no passwords are stored in flash memory so cryptanalysis will yield little data.

A.1.3 Power Supply

The Atmel AT90USB128 was configured to use +5 volts from the USB bus to UVCC, AVCC, DVCC, UVCON, VBUS, and JTAG. Though the voltage from the PC's USB port is highly regulated, bypass electrolytic capacitors were used to filter any ripple on V_{cc}. For bypassing ripple, 100 nF capacitors were placed as near as possible to V_{cc}. A LP3982 CMOS linear voltage regulator from National Semiconductor was used to provide a regulated +3.3 volts to the SPI bus via port B and the serial flash memory device. The LP3982 is an 8-Pin surface mount device with a 33 nF noise bypass capacitor to stabilize the voltage when the USB device is first plugged in. No debouncing circuitry was used for power-on. When the USB device is plugged into the port, software handles any debouncing. No reset switch was needed because the JTAG programming device can reset the microcontroller. In effect, the circuitry is minimal to reduce cost.

A.1.4 Schematic

USB Token Schematic



A.1.5 USB Firmware

The USB firmware is based on an Atmel mass storage application using SCSI commands developed as part of the AVR273 Application Note. Because most operating systems support mass storage with USB drivers, there would be no need for users to install any software or drivers to support the security token proposed. AVR273 states that the firmware is supported by all Microsoft OS from Windows® 98SE or later, Linux kernel 2.4 or later, and Mac OS 9/x or later. This should satisfy a majority of users.

The firmware is a bundled as part of the *USB flash microcontroller software suite* with a license to distribute the firmware only as part of an Atmel microcontroller product. For this reason, most of the firmware source was left intact with the addition of one module named cryptodev.c. This prototype module performs all the secure storage and data manipulation as outlined in this document. Cryptodev.c provides a layer between Atmel's firmware and the flash memory device. As such, in the event another microcontroller vendor is needed, this c module could be reused with another firmware bundle and minimal reprogramming. The Atmel module interaction with the prototype is illustrated in Figure 49. In the appendix is a list of all the modules that comprise the prototype's firmware.

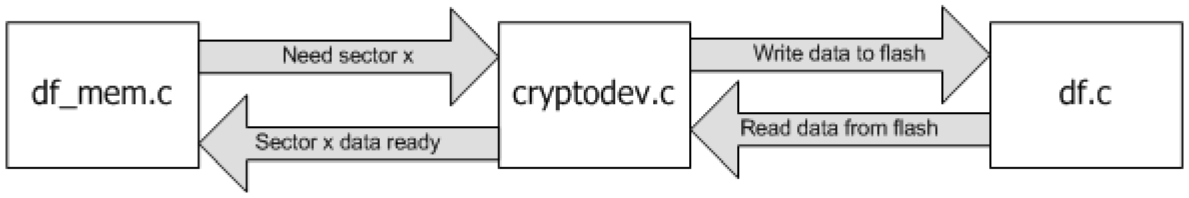
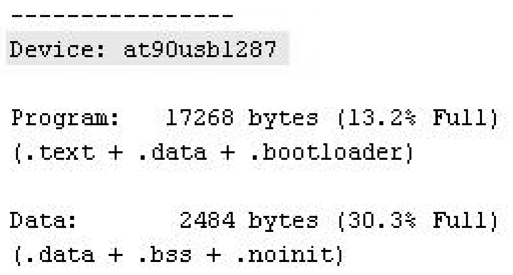


Figure 49: Data flow between Atmel module and prototype module.

A.1.6 Development Environment

All firmware was compiled using AVR Studio 4, version 4.15, IDE. The binary was downloaded to the USB device using the built-in JTAGICE mkII programmer module that the IDE supports. Debugging was also accomplished using the JTAGICE mkII module integrated in AVR Studio when the USB device was plugged into both the USB port and the JTAGICE mkII. The memory footprint for the code and data was 13.2% and 30.3% respectively (Figure 50). For this reason, the 64K MCU would be adequate.



```
----------------
Device: at90usb1287

Program:   17268 bytes (13.2% Full)
(.text + .data + .bootloader)

Data:       2484 bytes (30.3% Full)
(.data + .bss + .noinit)
```

Figure 50: Compiler output of firmware memory usage.

The client application was developed with Microsoft Visual Studio 2005 with service pack 1. For debugging purposes, the firmware module, cryptodev.c was incorporated into a USB device simulator. The USB device simulator was created to handle file I/O transfers of data to the embedded microcontroller and the flash unit.

A.2 System Communication

A.2.1 Byte order of data between client and device

Data type Position	**Byte Stream Offset**
Short data type used in X-Y values	
Low Byte	0
High Byte	1
Short data type used in command values	
Low Byte	1
High Byte	0
Integer representation used for nonce values and color values	
24-31	0
16-23	1
8-15	2
0-7	3

A.2.2 Commands/Responses and the Payload Summary

Command	**Description**
DEVICEID	A 16-byte unique value to identify the device to a server.
	Additionally, 180 text characters for the login graphical password are returned.
GETTEXT2 SAVETEXT1 SAVETEXT2	There is no GETTEXT1 because space was used during the DEVICEID command to send them. As with the the device login, the Web password requires 180 text values for the graphical password.

GETXY1A GETXY1B GETXY2A GETXY2B SAVEXY1A SAVEXY1B SAVEXY2A SAVEXY2B	There needs to be X-Y locations for each character in the graphical password. There are 180 characters giving 360 values. The order of data is the x-coordinate followed by the corresponding y-coordinate for each matching character index. Because these values are a short datatype, or two bytes, 720 bytes are needed to transmit all the values. However, the data payload allowed is 498 bytes. Therefore, two batches of coordinates are sent or received with 180 X-Y positions at a time.
GETIMAGE1 GETIMAGE2 SAVEIMAGE1 SAVEIMAGE2	Two images are used as the background for the graphical passwords: one for the login and one for the Web password. These images are limited to JPEGs of an allowable size. Since an image will span many payloads, image saving or retrieval must be done in repeated sequence. The image size is stuffed in front of the first payload. A state of what has been send or received and the image size determines when the transfer is complete.
GETLINKS SAVELINKS	The URL that the user considers a favorite for secure login can be saved or retrieved. The “http://www.” part is removed for brevity. An “*” before the URL means the HTTPS protocal and no “*” means the HTTP protocol. Each URL is separated by one white space.
CHANGEKEY	The 256-bit, AES key used to encrypt and decrypt data packets during file I/O is stored in the device. When the user logs into the device, this key is used for all sensitive data transmission.
GETSAMPLEPTS	The flash memory holds X-Y coordinates determined by the issuer of the device. These points are sent to the application to request the pixel color at those locations. There are 32 points sampled for a total of 64 values needed. Each value is a short datatype requiring 128 bytes of payload data.
SENDHASHPARTS	Once the user password comprising text, X-Y positions, and the sample color have been collected, these items are packaged and sent to the device so a calculated hash value for the password can be received. The text is one byte, the X-Y positions are 4 bytes and the color is 4 bytes. The maximum space is the payload area. The first byte of the payload tells the device how long is the password. From this, the device knows how long the data is in the payload.
GETHASH	If the MCU is finished mathematically computing the 32-byte value needed as a password, this value is sent back to the application.
GETFILECRC	For security, the portable executable CRC value for the application is stored in the device. The application will query this value to validate if the login was a success. An incorrect CRC should prevent further communication.

A.2.3 System Command and Response Definitions

Get the Device ID

Data Communicated

Item	Description
Device ID	16 unsigned bytes each byte can have the value of 0 to 255
Graphical Password Text	180 characters excluding special characters

Command

Command	Value	Device Action
CMD_DEVICEID	0x6255	Initialize state variables

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data
STATUS_FAILED	0x0000	System failed. Payload contains random garbage.

Response Payload

Offset	Length	Contents
14	16	16 byte ID unique to the USB device
30	180	All the text for the graphical password to login to the USB device

Get the X-Y Part 1 for USB and Web Login

Data Communicated

Item	Description
Password X-Y coordinates	360 short values in the form of X then Y for each point

Command

Command	Value	Description
CMD_GETXY1A	0x6256	Get the first part of the X-Y values for the USB login password.
CMD_GETXY2A	0x6357	Get the first part of the X-Y values for the Web password.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data
STATUS_FAILED	0x0000	System failed. Payload contains random garbage.

Response Payload

Offset	Length	Contents
14	360	360 unsigned bytes that represent 180 short values

Get the X-Y Part 2 for USB and Web Login

Data Communicated

Item	Description
Password X-Y coordinates	360 short values in the form of X then Y for each point

Command

Command	Value	Description
CMD_GETXY1B	0x6257	Get the last part of the X-Y values for the USB login password.
CMD_GETXY2B	0x6358	Get the last part of the X-Y values for the Web password.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data
STATUS_FAILED	0x0000	System failed. Payload contains random garbage.

Response Payload

Offset	Length	Contents
14	360	360 unsigned bytes that represent 180 short values

Get the Image for USB and Web Login

Data Communicated

Item	Description
Image raw data	Image data of the JPEG file.

Command

Command	Value	Description
CMD_GETIMAGE1	0x7070	Get the image for the USB login in blocks that fit the payload.
CMD_GETIMAGE2	0x7072	Get the image for the Web password in blocks that fit the payload.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status (STATUS_OK can be added to STATUS_FIRST)

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data
STATUS_FIRST	0x0001	Payload contains first part of image data
STATUS_FAILED	0x0000	System failed. Payload contains random garbage.

First Response Payload

Offset	Length	Contents
14	4	Total image size stored in the device
18	494	Image raw data

Second to N Response Payload

Offset	Length	Contents
14	14 to remainder	Raw image data up to the last image byte – following data is invalid

Get the Password Text for the Web Password

Data Communicated

Item	Description
Graphical Password Text	180 characters excluding special characters

Command

Command	Value	Description

CMD_GETTEXT2	0x6355	Get the text for the Web password.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	180	Password text

Save the X-Y Part 1 for USB and Web Login

Data Communicated

Item	Description
Password X-Y coordinates	360 short values in the form of X then Y for each point

Command

Command	Value	Description
CMD_SAVEXY1A	0x6258	Save the first part of the X-Y values for the USB login password.
CMD_SAVEXY2A	0x6359	Save the first part of the X-Y values for the Web password.

Command Payload

Offset	Length	Contents
14	360	360 unsigned bytes that represent 180 short values

Response Status (STATUS_OK can be added to STATUS_PPPOINTS)

Status	Value	Meaning
STATUS_OK	0x8000	No system errors
STATUS_FAILED	0x0000	System failed.
STATUS_PPPOINTS	0x0020	Text coordinates saved ok

Response Payload

Offset	Length	Contents

14	498	Randomized data

Save the X-Y Part 2 for USB and Web Login

Data Communicated

Item	Description
Password X-Y coordinates	360 short values in the form of X then Y for each point

Command

Command	Value	Description
CMD_SAVEXY1B	0x6257	Save the second part of the X-Y values for the USB login password.
CMD_SAVEXY2B	0x6358	Save the second part of the X-Y values for the Web password.

Command Payload

Offset	Length	Contents
14	360	360 unsigned bytes that represent 180 short values

Response Status (STATUS_OK can be added to STATUS_PPPOINTS)

Status	Value	Meaning
STATUS_OK	0x8000	No system errors
STATUS_FAILED	0x0000	System failed.
STATUS_PPPOINTS	0x0020	Text coordinates saved ok

Response Payload

Offset	Length	Contents
14	498	Randomized data

Save the Image for USB and Web Login

Data Communicated

Item	Description
Image raw data	Image data or the JPEG file.

Command

Command	Value	Description

CMD_SAVEIMAGE 1	0x7071	Save the image for the USB login in blocks that fit the payload.
CMD_SAVEIMAGE 2	0x7073	Save the image for the Web password in blocks that fit the payload.

First Command Payload

Offset	Length	Contents
14	4	Total image size to store. Will be checked by device for size limit.
18	494	JPEG raw data.

Second to N Command Payload

Offset	Length	Contents
14	14 to remainder	Raw image data up to the last image byte – following data is invalid

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Image saved ok.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	498	Randomized data

Save the Password Text for Login and Web

Data Communicated

Item	Description
Graphical Password Text	180 characters excluding special characters

Command

Command	Value	Description
CMD_SAVETEXT1	0x6260	Save the text for the login password.
CMD_SAVETEXT2	0x6356	Save the text for the Web password.

Command Payload

Offset	Length	Contents

14	180	Password text

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Saved text ok.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	498	Randomized data

Save the URL Links

Data Communicated

Item	Description
URL links	Several URLs seperated by a space. Must fit into the payload.

Command

Command	Value	Description
CMD_SAVELINKS	0x6969	Saves URL links in the device

Command Payload

Offset	Length	Contents
14	498	Text

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Saved text ok.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	498	Randomized data

Get the URL Links

Data Communicated

Item	Description
URL links	Several URLs seperated by a space. Must fit into the payload.

Command

Command	Value	Description
CMD_GETLINKS	0x7069	Retrieves a block of URL links in the device

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status

Status	Value	Meaning
STATUS_OK	0x8000	Payload contains valid data.
STATUS_FAILED	0x0000	System failed. Payload contains garbage data.

Response Payload

Offset	Length	Contents
14	498	Text

Change Login AES Key

Data Communicated

Item	Description
AES Key	32 unsigned chars of data formed from the Graphical password.

Command

Command	Value	Description
CMD_CHANGEKEY	0x7074	Change the login key in the device

Command Payload

Offset	Length	Contents
14	32	unsigned chars

Response Status (STATUS_OK can be added to STATUS_KEYOK)

Status	Value	Meaning
STATUS_OK	0x8000	No System error
STATUS_KEYOK	0x0002	AES key was saved and will be used for the next command
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	498	Randomized data

Send the Hash Parts for Return AES Key

Data Communicated

Item	Description
Values to create a Hash and Key	The complete payload reserved for data used by the MCU to form a hash

Command

Command	Value	Description
CMD_SENDHASHPARTS	0x7076	Process the data to find a 32-byte hash value

Command Payload

Offset	Length	Contents
14	1	How many items in the password. Limit is 70.
15	497	Color data from the image, text, and X-Y coordinates of the click points.

Response Status (STATUS_OK can be added to STATUS_HASHPWD)

Status	Value	Meaning
STATUS_OK	0x8000	No System error
STATUS_HASHPWD	0x0008	AES key was saved and will be used for the next command
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	498	Randomized data

Get Hash value for Web passwords

Data Communicated

Item	Description
Hash value	A 32-byte character string

Command

Command	Value	Description
CMD_GETHASH	0x7075	Get the computed a 32-byte hash value the MCU should have finished.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status (STATUS_OK can be added to STATUS_HASHPWD)

Status	Value	Meaning
STATUS_OK	0x8000	No System error. If only STATUS_OK then hash not ready
STATUS_HASHPWD	0x0008	Hash value computation finished and ready.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	32	32-byte hash value

Get the X-Y Points to Sample for Hash Value

Data Communicated

Item	Description
X-Y coordinates to sample	64 short values (32 X positions and 32 Y positions) in X then Y format

Command

Command	Value	Description
CMD_GETSAMPLEPTS	0x7077	Get 32 X-Y points stored in the device by the factory.

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status (STATUS_OK can be added to STATUS_HASHPWD)

Status	Value	Meaning
STATUS_OK	0x8000	No System error.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	128	64 short values or 128 bytes

Get the X-Y Points to Sample for Hash Value

Data Communicated

Item	Description
PE CRC	4 byte integer representing the CRC hash of the portable executable

Command

Command	Value	Description
CMD_GETFILECRC	0x7078	Get PE CRC

Command Payload

Offset	Length	Contents
14	498	Randomized data

Response Status (STATUS_OK can be added to STATUS_HASHPWD)

Status	Value	Meaning
STATUS_OK	0x8000	No System error.
STATUS_FAILED	0x0000	System failed.

Response Payload

Offset	Length	Contents
14	4	integer

A.3 DOS Header Reference

DOS Header found in PE format [82]

Data Type	Field	Definition
WORD	e_magic	Magic number
WORD	e_cblp	Bytes on last page of file
WORD	e_cp	Pages in file
WORD	e_crlc	Relocations
WORD	e_cparhdr	Size of header in paragraphs
WORD	e_minalloc	Minimum extra paragraphs needed
WORD	e_maxalloc	Maximum extra paragraphs needed
WORD	e_ss	Initial (relative) SS value
WORD	e_sp	Initial SP value
WORD	e_csum	Checksum
WORD	e_ip	Initial IP value
WORD	e_cs	Initial (relative) CS value
WORD	e_lfarlc	File address of relocation table
WORD	e_ovno	Overlay number
WORD	e_res[4]	Reserved words
WORD	e_oemid	OEM identifier (for e_oeminfo)
WORD	e_oeminfo	OEM information; e_oemid specific
WORD	e_res2[10]	Reserved words
LONG	e_lfanew	File address of new exe header

A.4 Key Strength and Command / Responses

Key strength during file I/O (Note all encryption is 256-bit Rijndael AES)

Encryption Key	**Command/Response**	
Weak AES key until communications are switched to higher encryption.	CMD_DEVICEID	
	CMD_GETXY1A	
	CMD_GETXY1B	
	CMD_GETIMAGE1	
Always uses strong AES key.	CMD_SAVEXY1A	CMD_SAVEXY1B
	CMD_SAVEXY2A	CMD_SAVEXY2B
	CMD_SAVETEXT1	CMD_SAVETEXT2
	CMD_GETTEXT2	CMD_GETXY2A
	CMD_GETXY2B	CMD_SAVELINKS
	CMD_GETLINKS	CMD_SAVEIMAGE1
	CMD_SAVEIMAGE2	CMD_GETIMAGE2
	CMD_CHANGEKEY	CMD_GETHASH
	CMD_SENDHASHPARTS	CMD_GETSAMPLEPTS
	CMD_GETFILECRC	

A.5 Sector Map and Description

Sector	Bytes	Contents
0	41,984	FAT for the file system of the device.
82	557,568	The client executable file.
1172	512	This is the input and output area. In the FAT table it looks like a file of 512 bytes. The client application saves encrypted data to the file for commands and the device stores encrypted data here as a response.
1173	512	Device ID: 16 bytes, text for first GP (180 bytes), Flags (4 bytes), AES key (32 bytes). This sector includes all data for secure device communication.
1174	512	This contains an array of seed values generated with a true RNG at the factory. The device can use these values to seed nonces for secure communication.
1175	512	This contains an array of text values that were generated by a true RNG at the factory. When the user picks items on the graphical password for Web login, a mapping to this text items will help password stretch and return a very random text string for authentication on the Web.
1176	512	The first 180 X-Y coordinates for the text for the first graphical password (360 bytes).
1177	512	The second 180 X-Y coordinates for the text for the first graphical password (360 bytes).
1178	512	The first 180 X-Y coordinates for the text for the second graphical password (360 bytes).
1179	512	The second 180 X-Y coordinates for the text for the second graphical password (360 bytes).
1180	512	The text for the second graphical password. The first graphical password used part of the sector that held the Device ID.
1181	84,480	These sectors store the first image as JPEG file.
1347	84,480	These sectors store the second image as JPEG file.
1513	512	This sector holds the URL favourites for a secure Web site.
1514	512	This is where bad file accesses are directed. For example, if the user tried to store many sectors of data onto the disk, all would be thrown into the trash bin.

A.6 Firmware Source Files

Firmware source files (AVR329: USB Firmware Architecture)

Source	Function	Created By
config.h	USB: SBC_VENDOR_ID, SBC_PRODUCT_ID, SBC_REVISION_ID	Atmel
df.c	Contains the low-level dataflash routines. Has the FAT format table. (modified slightly to add a new memory routine)	Atmel Author
df_mem.c	Contains the interface routines of Data Flash memory. (modified slightly to add a new memory routine)	Atmel Author
dower_drv.c	Contains the Power management driver routines. (software debouncing)	Atmel
scheduler.c	Manages the routine to call the tasks. Each task will be called following the order specified in conf_scheduler.h	Atmel
scsi_decoder.c	Contains routines to decode and to manage the SCSI commands	Atmel
spi_lib.c	Provides a minimal function set for the SPI.	Atmel
storage_task.c	Manages the mass storage task.	Atmel
usb_descriptors.c	Parameters/values of the enumeration descriptor structures defined inusb_descriptor.h. This file is important for the user to make his own application	Atmel
usb_device_task.c	Manages the mass storage task.	Atmel
usb_drv.c	Interface between the user and the USB hardware, it contains all the USB drivers routines	Atmel
usb_specific_request.c	Contains the specific request decoding for enumeration process	Atmel
usb_standard_request.c	USB endpoint 0 management routines corresponding to the standard enumeration process.	Atmel
usb_task.c	Performs all USB requests (Standard and Specific). It manages the USB enumeration process and all asynchronous events (Suspend, Resume, Reset, Wake up...)	Atmel
wdt_driv.c	Contains the Watchdog low level driver definition.	Atmel
ctrl_access.c	Access Memory Control (not important refer to cryptodev.c)	Atmel
aes.c	AES encryption algorithm based on: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf	Atmel Finland
cryptodev.c	The driver that handles all the security implementations at the file sector level.	Author
Sha256.c	SHA-256 Security Hash module	GNU: Christophe Devine

A.7 Quality Function Matrix

A.7.1 Financial Costs

What effort and financial cost are users willing to expend to implement a better solution? What are users willing to pay for added security? Should users or the authenticating institutions incur the cost - or both? If the system were strictly deployed on the server, then authentication costs might be borne by companies than clients. If the solution is PC or device based, would clients pay for it or would institutions pay to create and deploy it to clients? What materials are involved and man-hours needed to create and deploy the solution? How will the solution be deployed: in software or in a physical device. What liabilities will this solution incur if it fails? In short, what is the value of the assets it protects? For example, what financial losses could be lost relative to the cost of the solution? What is the cost to engineer a very trusted system? For example, will engineers and manufacturers need thorough, high-level background screening and clearance? Cost is usually very tightly coupled to risk aversion: as the risk of failure goes down, the cost will go up. How quickly the cost will increase when there are further security features is a function of quality and design.

A.7.2 Usability

It is clear that statistical evidence points to large financial losses by both institutions and consumers due to phishing and maybe less so by password theft. However, how much is the effort worth for users to adapt to a better system? The current system in use today is a text-based password system. Users are very familiar with a user name and password scheme and most secure Web sites employ this means of authentication. Can users and institutions adapt easily to a different system and if so, to what degree? Can another authentication system be easy to learn, remember and feel secure? Any new system would also suffer if it were too slow or require the user to do too many tasks. Are there any installation issues or complexities? Those systems that require a user to have enhanced memory or handle complex tasks may be a problem for individuals with less education or reduced cognition. If some users are visually impaired or color blind, would they be able to use the system? How does the solution handle international users and languages? The success of any system is not only if it “really” is secure but if users “feel” secure. If the user can’t trust the company, the documentation, the deployment, or the degree that they might be attacked, they might feel it is not a useable solution.

A.7.3 Functionality

Though security may seem to be the main feature above all, there is a continuum of required security relative to the existing text-based password system. On one end, if the security of a solution is less or equal to the current text-based password system, there is no compelling reason to implement one. On the extreme end of the continuum, the usability and cost features may become an aggregate negative weight on the total solution evaluated. When analyzing the current text-based password system, it should be realized that passwords need to be highly randomized and hard to guess or predict. The password space must be large enough to ensure that the possibility of a correct guess is low enough. However, there are theoretical password spaces and a practical password spaces. For example, with text-based passwords, users may pick easy names even if 8 characters are used. If randomized, long passwords are not enforced, the practical password space may be much smaller than the theoretical space. Unarguably, any solution that can enforce a better password space is required. Additionally, how effective is any solution against Trojans and specifically against key and mouse loggers? Is there any protection against pirated or bad URLs? In finding the right continuum of security solutions that work, one should measure the efficacy in preventing attacks in metrics that can relate to a risk factor. Since cost is a concern, the risk of any solution failing should be associated with the cost of production.

It is important to know if the solution will be Web deployed, installed as an executable, or run on a separate device. Each of these deployments carries their own risk and usability issues. A Web deployment would have to solve the MITM attack or be able to assess that risk. An installed binary could be attacked by viruses. Likewise, any hardware device can have vulnerabilities due to poor understanding of the device or device failure. In essence, deployment definitions help define a security risk, cost or usably.

A.7.4 Maintainability

The ability to setup a test case with measurable, distinct metrics verifies the design and efficacy. If the source code is obfuscating the logic, it makes the solution hard to evaluate strengths and quantify risks. For example, what encryption is being executed and what tokens are inputs or outputs? If all possible inputs and outputs can't be analyzed, then that solution has a higher potential risk.

If the software code is not analyzable, then metrics to measure branch and conditional coverable would be incomplete. If data communication used non-standard protocols, then there must be an oracle that can adequately verify it. Without a testable design, there is less likelihood to properly assess the risk of a better solution.

A.7.5 Reliability

Tantamount to a secure solution is how reliable it is in executing its design. How many faults does it have over a period of time? If the solution fails, could there be a security breach? Would the user find the failure a burden and abandon the solution? How mature is the technology it uses? A mature technology has more history to better assess risk. How effective is the solution against attack?

A.7.6 Efficiency

How long will the new solution increase or decrease the login time? A longer login time might increase the change of shoulder surfing. Will the user recall the password better which might reduce login errors? Are there any extra setup steps such as plugging in a device or downloading an application? Will the user's productivity be affected by installation and activation issues?

A.7.7 Portability

In viewing portability, a question to ask might be can the solution work on different operating systems? It is possible to use this solution across different languages and cultures? How will does this solution conform to other standards? For example, in the United States, one guideline for authentication is the *Electronic Authentication Guideline* written by the National Institution of Standards and Technology.

A.7.8 Requirements

In using the SQM to define requirements and evaluating a better, secure Web authentication system, there will be the need to weight some characteristics more than others. To do this, it would be advisable to poll industry leaders and end-users to determine an appropriate weight. If only cost and security were the only two important characteristics, a simple ratio of two metric aggregate scores could be used to determine a better solution. However, there are other characteristics such as usability and efficiency, to name a few, that would need to be factored into an overall score. Each characteristic is itself an aggregation of sub scores. For a security score, the score would depend on the risk of attack. For a cost score, the financial cost of the assets that can be breached relative to the total retail cost of the solution. Using a scale of 1 to 100, a rollup score of all the attributes according to weight can determine the better solution.

Table 1: Characteristic requirements by weight (ISO/IEC 9126-1)

Characteristic	Weight	Description
Functionality	0.50	This includes the security attributes that define risk. A score would be based on the risk of attack.
Cost	0.30	This is the score based on the cost of the assets protected to the retail cost of the solution. This score could be a simple ratio or a weighted average.
Reliability	0.10	How failsafe the device will be over time. The number of faults or errors measured. The risk of a security breach if the device fails.
Usability	0.025	The score related to login time, the time it takes to learn the system and the overall feeling of the whole system. The percent of users that can use the system (level of education or visual disabilities)
Maintainability	0.025	The ability to patch the system to fix security holes easily or upgrade the system to meet further security requirements.
Efficiency	0.025	The time it takes to start the application and authenticate the user.
Portability	0.025	The ability to work on other operating systems, cultures and languages.

A.7.9 Metric Definitions

Table 2: Financial Average Costs

Characteristic	Metrics	Description
Manufacturing	1. Equipment & materials	The amount spent on manufacturing.
Development	2. Man-hours engineering	The development cost is needed.
Deployment	3. Shipping & installing	The deployment cost.
End-user	4. Retail cost	The cost the end user might pay.
Assets	5. Replacement cost	The cost of the assets if stolen due to a login breach.

Table 3: Usability

Characteristic	Metrics	Description
Understandability	6. Learning time during practice session.	Time is used to measure the difficulty to learn the new solution during practice login sessions.
Learnability	7. Number errors in practice session.	The number of login errors during a practice session.
Operability	8. Errors per logins. 9. Setup time.	When the user logs in after a period of time, the errors are recorded. The time to setup the application is also essential.
Attractiveness	10. Scale of satisfaction.	Users can be polled on a 1 to 5 questionnaire about their experience and overall feeling of the system.

Table 4: Functionality

Characteristic	Metrics	Description
Suitability	11. Risk level of attack.	A level given to the possibility of being broken by an attacker.
Accurateness	12. Number of false positive/negatives	A measure of allowing the correct user to authenticate.
Interoperability	13. Works with current text-password system. 14. Number auth. protocols it supports.	Can it work with the previous text-based system? What protocols does it support?
Compliance	15. Security standards using NIST 800-63.	Using the NIST level system. 1 to 4 security levels with 4 the highest.
Security	**Security Token generation:** What you have 16. Stored token is password protected 17. Stored token requires activation 18. Stored tokens encrypted 19. Token issued by trusted authority 20. Token deployed by trusted method 21. User verified before token handoff 22 Token lifetime 23. Tamper detection and destruction 24. Token can be copied 25. Token can be extracted What you know 26. Easy to communicate accidentally 27. Shared secret to identify the server 28. Password entropy > 10^{12} 29. Enforces hard to guess password 30. Shoulder surfing resistant 31. Key-logger resistant 32. Mouse-logger resistant 33. Activation password needed 34. Minimum password length>8 objects 35. Enforces minimum password length 36. Input time required < 5 seconds 37. Brute force attack possible What you are 38. False positives possible 39. Protection against loss of privacy 40. Protection against loss of body part **Security handshake:** Uses two factor authentication 41. What you know + what you have 42. What you know + what you are 43. Validates by password 44. Validates by encryption key 45. Maintains session with secure token 46. One-time encryption key used 47. Symmetric or asymmetric keys 48. Key exchanges during session 49. Key changes after login and stored	For security risks and scores, if the metric is a binary value, then 5 or 0 can be used. For ranges of metric results, then 0 to 1 with values ranging from 0.001 to 1.000.

	50. Verified public keys by a CA 51. Replay attack risk 52. Impersonation risk **Data exchange:** 53. Key used is 256 bit AES or 1024 RSA 54. IO encryption 55. Randomized nonce used in encryption 56. Hijacking session risk 57. Eavesdropping risk 58. Man-in-the-middle risk	

Table 5: Maintainability

Characteristic	**Metrics**	**Description**
Analyzability	59. Code reviewed	The source code was code reviewed by a third-party security firm.
Changeability	60. Can a security patch be applied to fix a weakness? 61. Time to implement a patch once a security weakness is discovered.	Is there a way to fix any security flaws? If so, how long does it take?
Stability	62. How many flaws are found with each upgrade?	If the design has changed, how many bugs are reported for that version.
Testability	63. Can the system be verified when a change occurs?	If the system submits encrypted data, this may be hard to do.

Table 6: Reliability

Characteristic	**Metrics**	**Description**
Maturity	64. Number system failures. 65. Number of security failures.	A measure of all failures. A measure of security failures.
Fault tolerance	66. Can the system lock out the user upon failure?	Any unrecoverable failure should lock the user from the authentication system.
Recoverability	67. Does system recovery return the system to a secure one?	If the system can recover, can it guarantee a return to a secure state?

Table 7: Efficiency

Characteristic	**Metrics**	**Description**
Time behavior	68. Login time	How much time does it take on average to authenticate?
Resource behavior	69. Requires keyboard IO 70. Requires mouse IO 71. Requires disk storage 72. Requires touch screen 73. Requires sound 74. Requires camera 75. Requires biometric scan	Certain IO ports are vulnerable to attacks. Also, how the application communicates with any IO ports is a point of risk. Does the solution communicate with IO in encrypted format (see security)? Also, the more complex IO, the more risk of failure. A biometric device can be more complex than a mouse or keyboard.

Table 8: Portability

Characteristic	Metrics	Description
Adaptability	76. How many OS platforms 77. How many Web browsers 78. Does not requires other software	Windows, Mac, Linux? Microsoft, Mozilla, Safari Should other software be installed before this?
Installability	79. Any special device drivers needed?	If proprietary drivers are needed, it may fail on some OS versions and libraries. Also, a more proprietary driver could mean more risk than a proven driver.
Conformance	80. Conforms to OS install policies?	For example, Microsoft wants any device drivers to be registered and signed by Microsoft.

A.7.10 Base Measures

In order to find a uniform way to rank a better solution, sundry metric values must be mapped to a reasonable, uniform scale. For this scale, a range of zero to five will be chosen. Zero denotes that a metric does not apply. One denotes the poorest score and five the best score. For example, a security attribute score of four would indicate low risk high security authentication. Additionally, a perfect score, five, would be rare. Any metric that gives a range of values should be divided into five parts to give a fair weight to assess the scaled value. However, there may be times when simple division will not weight properly, such as logarithmic ranges. In non-linear ranges, the scale can be moved arbitrarily to a best fit.

Using any scale with a minimum and maximum value implies the measured boundaries must be known in the whole population. However, in many cases, this is not a known. For example, with cost, it is not known what the cheapest or must expensive cost could ever be. The best way to determine a scale would be relative to another in the same sample set. That is, the scale would denote how many multiples of measured units one solution is in reference to the extreme solution in that set. A tie would mean that the same score is used, and if only two were in the set, then they could both be given a one. A wider range in measured values could signal a statistically significant difference but a minor variation from the extreme may indicate no significant difference.

Table 9: Metric score determination chart

Metric #	Value	Formula	Value to Score mapping
1-4	percent	attribute cost/ Largest cost in comparison = % cost	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest cost of all. NOTE: Up to that percent.
5	percent	Retail cost /Asset replacement cost	5=1%, 4 = 10%, 3= 20%, 2=30%, 1 = 100% where 1 is highest cost of all
6	percent	attribute ave. time/ Largest ave. time in comparison = % time	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest time of all. NOTE: Up to that percent.
7	percent	attribute ave. errors/ Largest ave. errors in comparison = % errors	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest cost of all. NOTE: Up to that percent.
8	percent	attribute ave. errors/ Largest ave. errors in comparison = % errors	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest cost of all. NOTE: Up to that percent.
9	percent	attribute ave. time/ Largest ave. time in comparison = % time	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest time of all. NOTE: Up to that percent.
10	feeling	Average of a survey	5=great, 4=good, 3=no feeling, 2=bad, 1=hate
11	risk	Based on the NIST 800-63 and relative to text-based passwords	5=very low, 4=low, 3=no improvement, 2=slightly worse, 1=much worse
12	percent	Average % of false positive ID which allows successful login.	5=0%, 4=5%, 3=10%, 2=20%, 1=100%
13	bool	Works with text-based passwords?	5=yes, 0=no
14	number	How many security protocols?	5=>4, 4=>2, 3= only one
15	level	Level of security compliance	5=4,3=2,2=1,1=0
16-60	bool	Passes or fails	5=yes, 0=no
61	percent	attribute ave. time/ Largest ave. time in comparison = % time	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest time of all. NOTE: Up to that percent.
62	percent	attribute ave. flaws/ Largest ave. flaws in comparison = % flaws	5=5%, 4 = 10%, 3= 20%, 2=30%, 1 = 100% where 1 is highest cost of all. NOTE: Up to that percent.
63	bool	Passes or fails	5=yes, 0=no
64-65	percent	attribute ave. failures/ Largest ave. failures in comparison = % failures	5=5%, 4 = 10%, 3= 20%, 2=30%, 1 = 100% where 1 is highest cost of all. NOTE: Up to that percent.
66-67	bool	Passes or fails	5=yes, 0=no
68	percent	attribute ave. time/ Largest ave. time in comparison = % time	5=20%, 4 = 40%, 3= 60%, 2=80%, 1 = 100% where 1 is highest time of all. NOTE: Up to that percent.
69-75	bool	Any IO is a risk. To be fair, assign 5 to no IO and 1 to having IO	5=no, 1=yes
76-77	number	The more the better	5=>3, 4=2, 1=1
78-80	bool	Passes or fails	5=yes, 0=no