Monthly Report 2

Wu

Senior Design project

Caleb Schwind and Michael Pritchard

Distributed Authentication System

with Page-wise Encrypted File Access

Supervised by Dr. Chwan-Hwa Wu

# Abstract

Information security, especially in today’s highly connected and digitized technological environment, is an incredibly complex and difficult task. The increasing array of lethal attack vectors employable by assailants require equally sophisticated defense mechanisms that must also balance user convenience and accessibility. Given that many systems are vulnerable to single points-of-failure, it becomes absolutely essential that each and every aspect of a system is hardened against assault.

In recent history zero-day, memory-based exploits and in-time memory scraping attacks have been shown to be incredibly lethal with regard to existing information security defenses. An attacker does not need to compromise an authentication/authorization system -- he/she merely needs to wait until authorized users have been authorized by the system and accessed sensitive information. Once the data is in memory (in plain text), the assailant can capture the information without the need to bypass any security measures. It was precisely this type of attack that led to the widespread loss of credit card and other transactional information from Target stores around the nation.

The motivation behind this project stems from a desire to construct a system that repels zero-day, mutated attack vectors, provides no single point-of-failure for complete system compromise, and mitigates the potential losses due to in-time memory scraping attacks and other potential memory-based exploits.

A synchronous, multi-device logon procedure using specially designed software keyboards coupled with dynamic obfuscation techniques ensures that the user’s password cannot be reverse-engineered by malicious parties, regardless of whether the system as a whole has been compromised. Pre-negotiated information held on external storage devices prevent device spoofing, while a built-in suicide procedure handedly defeats brute force and side channel attacks. Finally, a distributed, page-wise file protection scheme protects data not only from brute-force decryption, but also minimizes both the window in which a memory-scraping attack could occur and the data exposed at any given time.

Table of Contents

[Abstract 2](#_Toc386470180)

[1. Introduction 3](#_Toc386470181)

[2. Design 3](#_Toc386470182)

[2.1 Design Overview 3](#_Toc386470183)

[2.2 Design Detail 6](#_Toc386470184)

[2.2.1 Registration Procedure 6](#_Toc386470185)

[2.2.2 Login Procedure 8](#_Toc386470186)

[2.2.3 Mutual Authentication 9](#_Toc386470187)

[2.2.4 Suicide Procedure 11](#_Toc386470188)

[2.2.5 Feedback Code 11](#_Toc386470189)

[2.2.6 AES GCM Encryption 14](#_Toc386470190)

[2.2.7 File Splitting and Merging 14](#_Toc386470191)

[2.2.8 Document Splitting 18](#_Toc386470192)

[2.2.8 Software Keyboard 22](#_Toc386470193)

[3. Implementation Overview 23](#_Toc386470194)

[4. Demonstration 23](#_Toc386470195)

[5. Results 29](#_Toc386470196)

[5.1 Performance 29](#_Toc386470197)

[5.2 Security Analysis 30](#_Toc386470198)

[6. Conclusion 30](#_Toc386470199)

[7. Works Cited 30](#_Toc386470200)

# 1. Introduction

In addition to several basic security measures commonly found in file protection systems, this design incorporates two key elements that serve to differentiate it: the authentication/authorization procedure, and the page-wise file protection scheme.

During the registration process, a user must enter his/her password across multiple devices – the order in which the password sequences are entered is left up to the user; however, the sequences must be entered in the same manner on subsequent logins. In order to successfully log in, the user must also possess the required external storage devices that were connected during registration. Login-specific information is contained on the devices and updated with each successful login. To protect against brute-force and side-channel attacks following the physical theft of external storage devices, a suicide procedure has been implemented. After a predetermined number of incorrect logon attempts, the information on the external devices is destroyed, making successful authentication impossible. The data can be retrieved by an authorized user from cloud storage.

The input of the password itself is carried out via a software keyboard hardened against screen capture and keylogging attacks. The keyboard utilizes character groupings in place of traditional 1:1 character to button mappings. Combined with dynamic input obfuscation, this guarantees that the user’s true password can never be derived by an attacker. Intra-session key location scrambling coupled with inter-session key grouping scrambling render screen capture attacks futile.

Files protected using this system go through several processes. First, documents are split page-wise. These pages are then divided into bulk and piece files, which are encrypted and distributed amongst the clients and server. Documents are accessed one page at a time, and only the accessed page is unprotected and loaded into memory; furthermore, the page is immediately re-protected upon closing. While not a perfect solution to in-time memory scraping attacks, this does limit what data is exposed at any given time.

# 2. Design

## 2.1 Design Overview

This design requires the use of three devices for file protection: a server, an Android phone/tablet, and a PC. In addition to these devices a USB and SD card will be needed for the PC and Android device respectively. The only information the user will be required to know is the IP address of the server, the password, and the order in which to input the password on the PC and Android devices. Upon five unsuccessful password attempts the program will initiate a suicide procedure to prevent attackers from accessing the data. Figure 1 shows how to login to the system with the password set to “War\_Eagle” where the input ordering is “War\_” on the PC, “Eagl” on the Android, and “e” on the PC.

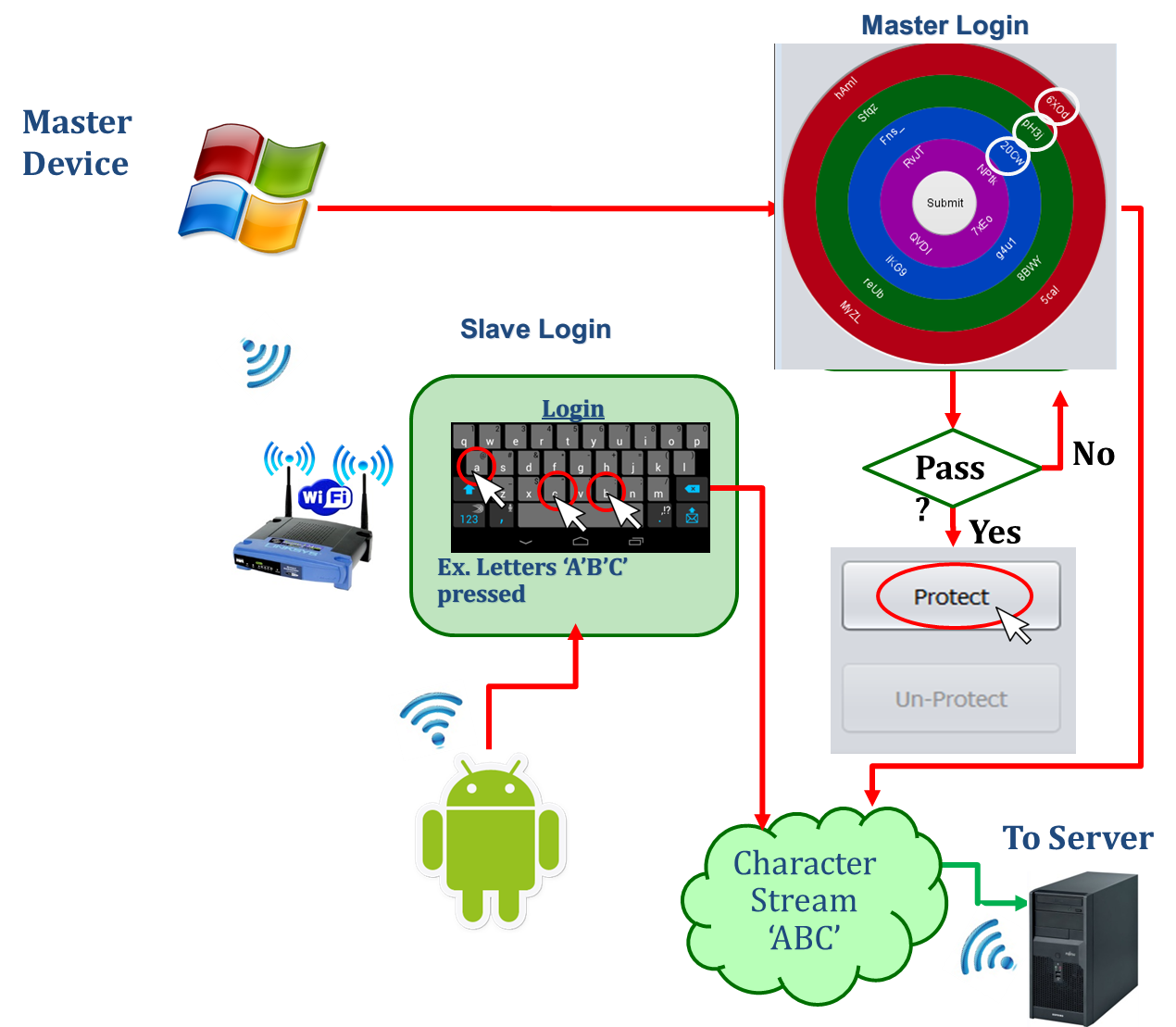


Figure 1. Login procedure diagram

The file security system is split into two major components, Protection and Unprotection. The protection system block diagram is shown in Figure 2 and the unprotection system block diagram is shown in Figure 3.

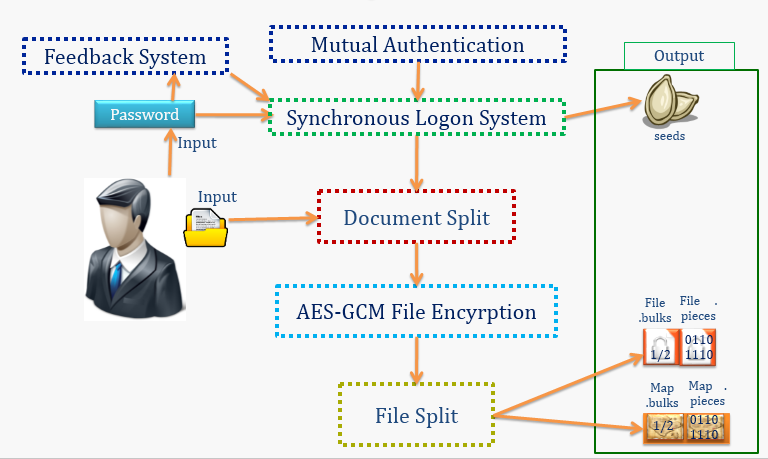


Figure 2. Protection block diagram

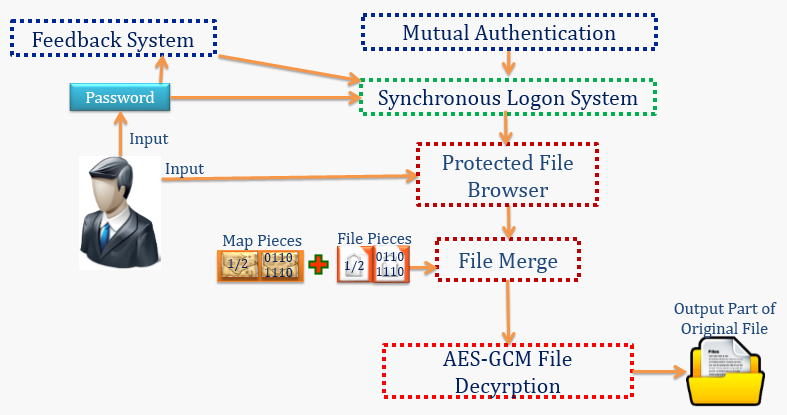


Figure 3. Unprotection block diagram

The design is split into 7 main components:

* Mutual Authentication:

Create a secure communication channel between each device. Each communication channel is to have a shared secret that is used to encrypt the data being sent between the two devices. Part of the shared secret is derived from user input.

* Feedback System:

Java code that is run from external storage. The feedback code is used hash each of the key presses with the previous hash output. The first key pressed will be hashed with a random number; the random numbers are inserted using metamorphosis code. This design uses the feedback code for the first three key presses only. The final output is used to update the mapping tables for the software keyboards. Since these codes are stored on external storage they are started using a batch file and then Java RMI/LipeRMI is used to access the methods remotely.

* Synchronous Logon System:

Use both the client PC and Android phone/tablet as password input devices. The client PC utilizes a software keyboard that randomizes the key locations, additionally each key will be mapped to a random string value via mapping table. The Android device uses a stock Android keyboard that requires the user to translate the desired key grouping to the Android key the user needs to click by looking at the PC program. Each key press is obfuscated (via mapping table) with two seeds, encrypted (via AES-256 GCM) with two keys, and then both results are sent to the server. After the server receives all of the key presses the two passwords (based on the two different seeds) will be reassembled. One of the obfuscated encrypted passwords will be used for the current login while the other will be for the next login. The server authenticates the password by decrypting the entered password and the stored password and then compares the two obfuscated passwords to see if they match and the result is sent back to the PC and Android.

* Document Split:

Splits PDF documents page-wise using the Apache PDFBox library [1]. The documents are split page-wise so that when the user wants to view a document they can only have one page of the document, in its unprotected form, in memory. Each document and the number of pages of that document are added to a map file. This reduces the amount of information lost to an in-time memory scraping attack. This component only supports PDF documents, but other file formats can be added by including more libraries.

* AES-GCM encryption/decryption:

AES-256 GCM was used as the encryption/decryption method for this design [2]. This design uses the GCM (Galois/Counter Mode) to add authentication and ensure confidentiality of the AES encryption [3]. All of the keys and IVs are randomly generated using Java’s Secure Random.

* File Splitting and Merging system:

Splits the encrypted files into two separate files, pieces and bulks, based on random numbers. Index files are used to store the keys, file names, file paths, and random numbers. The index files are also encrypted and split, but the random numbers are stored in a text file instead of another index file.

* Protected File Browser:

Used to view the protected files and view a selected page of the desired document. The file browser uses a map file to keep track of the document name, location, and number of pages of the document. The protected file browser displays each document along with the number of pages of that document.

## 2.2 Design Detail

### 2.2.1 Registration Procedure

The first step for the use of this program is user registration. This procedure mainly involves username and password registration. First the user will be required to enter a username and password into the software keyboard, shown in Figure 4. The username field is entered in using the hardware keyboard and the password is entered using the software keyboard.

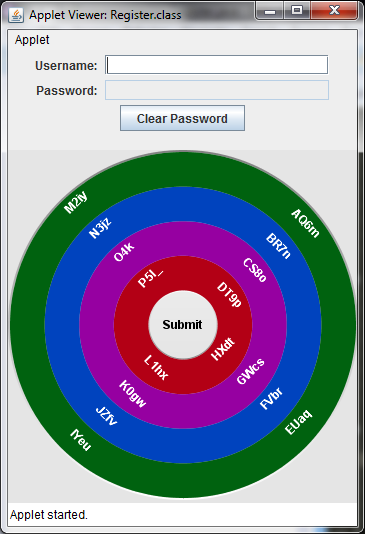


Figure 4. Software Keyboard

The password will follow the format shown in Figure 5. The user will enter the password into the PC and Android devices in their preferred order during registration and the order in which the password needs to be entered for future logins will be established here. The combined passwords are used for authentication with the server.

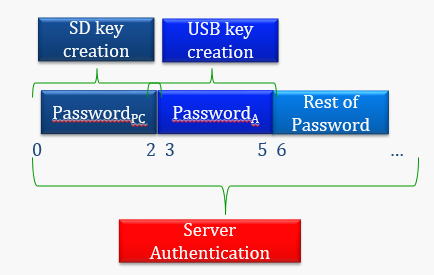


Figure 5. Password setup

The procedure the system follows for registering the password is shown in Figure 6. When the password is entered into the system the seeds (seed1) and keys (key1) for the next login will be saved (the keys will be encrypted) on the external storage for the PC and Android. The initial seed is used to generate the key mapping table, which is used to obfuscate the password. After the password entry is complete the password will be encrypted with the initial key (key1), which is generated randomly. Finally the obfuscated encrypted password is sent to the server where it is stored for the first login. The key used to encrypt the password will not be sent to the server until the 1st login.

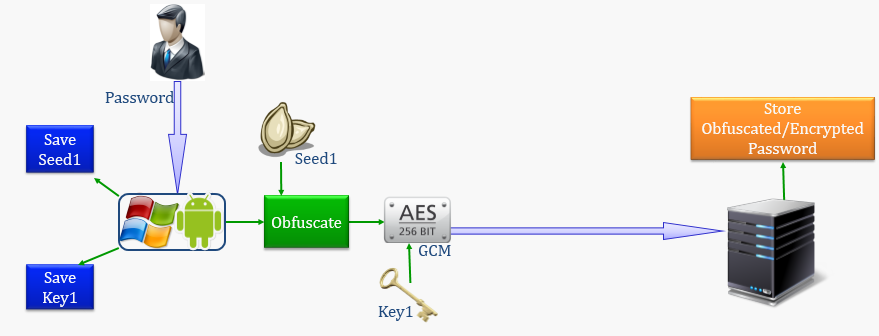


Figure 6. Registering password

### 2.2.2 Login Procedure

The login procedure will involve the user entering their username and password just as was done during the registration procedure. The system will follow the procedure outlined in Figure 7. This procedure involves using the seed (seed1) generated at registration to obfuscate the password while simultaneously using a new seed (seed2) to obfuscate the password for the next login. The key (key1) created during registration will be used to encrypt the obfuscated password and a new key (key2) will be generated and used to encrypt the password that was obfuscated using the new seed. Both of the obfuscated passwords and the key generated at login will be sent to the server. The server will save the password that was obfuscated using the new seed. The server will use the key generated at registration to decrypt the password (obfuscated using registration seed) and the password that was stored in registration. The server will then compare the two obfuscated passwords to authenticate the login. The server will finally send a response message back to the client devices (PC and Android device).

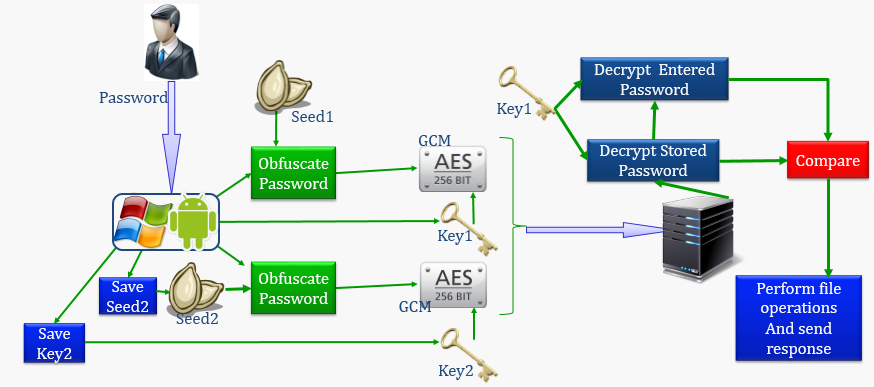


Figure 7. 1st login procedure

The registration and 1st login procedures are special cases and don’t require the deletion of any files. The procedure for the Nth login is detailed in Figure 8. The procedure for the Nth login is very similar to the 1st login procedure, except upon successful login the N-1 seed and the N-1 key will be deleted from the external storage devices. Also, upon successful login, the obfuscated encrypted password that was used for the current login will be deleted.

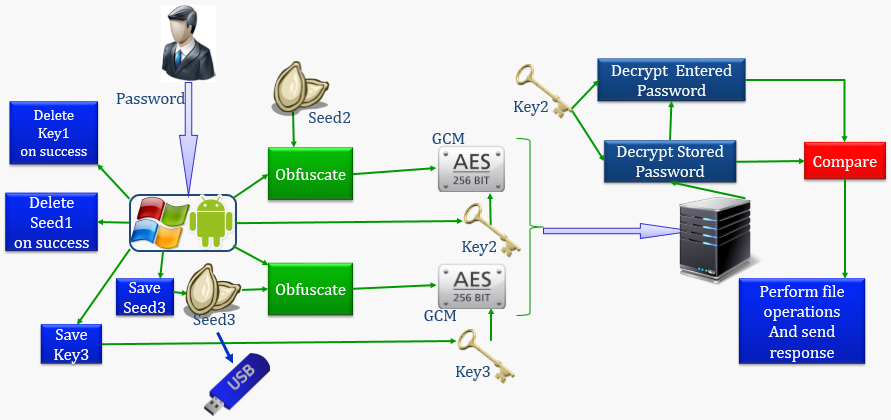


Figure 8. Nth login procedure

The server compares the two obfuscated passwords (stored and entered) after decryption. If the entered password was valid the server deletes the stored obfuscated password, saves the encrypted entered obfuscated password, deletes the keys used to encrypt/decrypt both passwords, and then sends a successful authentication message to the PC/Android. If the entered password was invalid the server deletes the entered obfuscated password, the received encryption/decryption keys, and then sends a failed authentication message to the PC/Android. If the PC/Android receives a message stating successful authentication (valid password) they will delete the latest seeds and keys, but leave the stored seeds and keys alone. If the PC/Android receives a message stating failed authentication (invalid password) then the oldest seeds and keys will be deleted, the latest seeds and keys will be stored, and the filenames of the seeds and keys will have their index values adjusted (0 = current login, 1 = next login). Figure 9 shows the flow chart detailing this procedure.

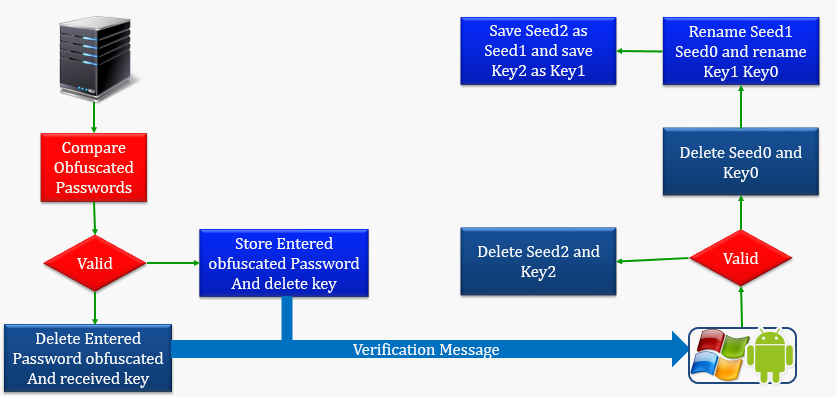


Figure 9. Validation flow chart w/ file operations (keys, seeds, and passwords)

### 2.2.3 Mutual Authentication

The basic communication channel will use the User Datagram Protocol (UDP), which is a lightweight communication protocol [4]. Each communication channel will be encrypted using a shared session key, shown in Figure 10. These keys will be used to encrypt all data that is sent over these communication channels. Public key encryption was a consideration for this portion for the asymmetry it provides but was decided against because the server would need to hold the private keys and since the server would have access to the public keys the server could effectively disguise itself as the client devices [6].

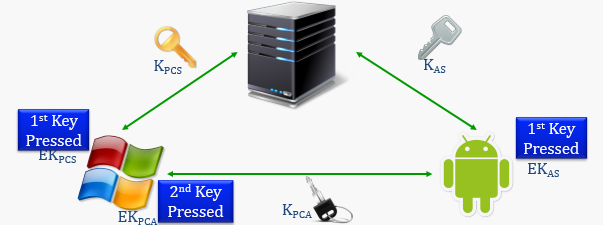


Figure 10. Keys for encrypting communication tables

The first step for generating this key will be to exchange seeds and XOR the seeds together. These seeds will be concatenated with the output of the 1st key press and 256 bits of this data will be chosen to create the AES encryption/decryption key [2]. The first round of encryption will just encrypt a string of zeroes. The cipher text output will then be encrypted again to produce the shared encryption key. This shared encryption key will be sent to the other device on the communication channel. Figure 11 details the procedure for generating the communication channel encryption keys.

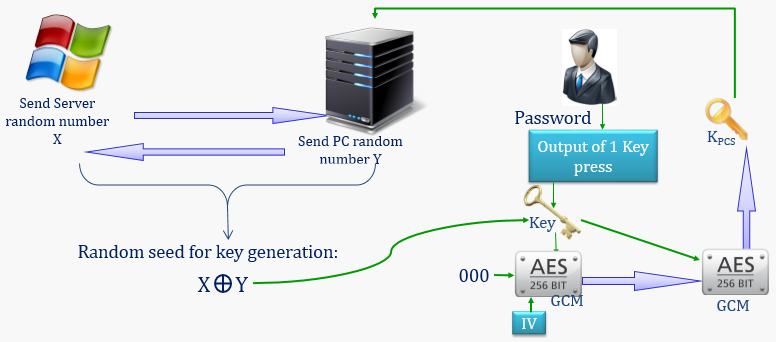


Figure 11. Generation of the communication channel encryption key from both PC and Android

### 2.2.4 Suicide Procedure

The suicide procedure will be activated if there are five incorrect passwords entered into the system. This procedure is outlined in Figure 12. Each failed attempt to login will result in the number of tries being incremented. When the number of tries is equal to six the suicide procedure will be activated. The suicide procedure will convert all of the seeds and keys to strings of zeroes. There will be no messages indicating that the suicide procedure was activated, so that the attacker can keep trying to enter in passwords and no possible combination will result in a successful logon. The seeds and keys will be backed up on a cloud service, so when the real user needs to login they can just pull this data off the cloud service and put them on the external storage devices to retrieve their data. Currently the only cloud service supported is DropBox [7].

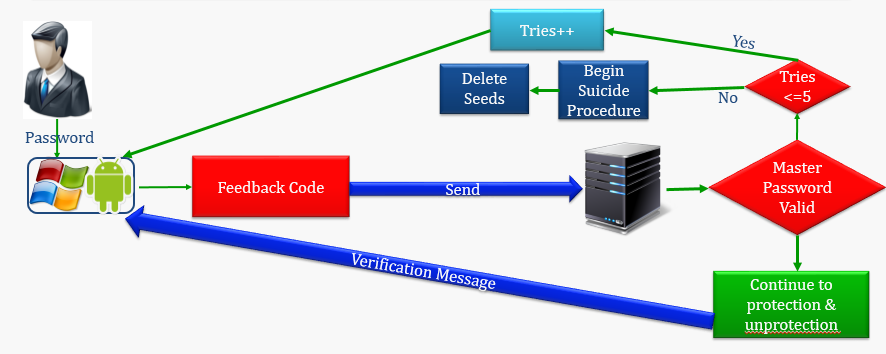


Figure 12. Suicide procedure overview

### 2.2.5 Feedback Code

The feedback code is used to hash, using SHA-256, the random strings of the first three button presses (C0, C1, and C2) with a random number (Rx) in an iterative manner [8]. SHA-256 is the secure hash standard which is a one way function to produce a condensed representation of the input data and output 256 bits of data (in the case of SHA-256) [8]. This string will be used as a seed (Sx’) to generate the master table that will be used to obfuscate the rest of the password. The first three buttons pressed are obfuscated using the initial seed (Sx). Figure 13 shows the procedure for generating the seed that is used to create the master table, where mapping function is the mapping table created from the initial seed.

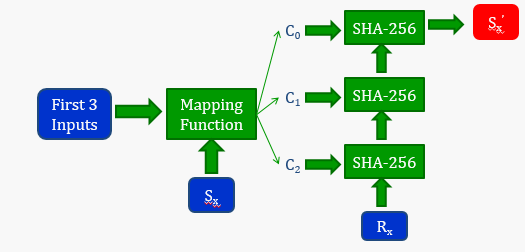


Figure 13. Feedback loop

The random number will be inserted into the feedback code using metamorphosis code, which just opens the feedback code file writes the random number in and closes. When the feedback code has received the strings from the first three button presses the feedback code will return the seed used to generate the new master tables and the running feedback code will close. The metamorphosis code will be called again to insert a string of zeroes into the random number variables of the feedback code. Both pieces of code will be started by executing a batch file. The main program will call the feedback and metamorphic methods through the use of Java RMI (Remote Method Invocation) [10]. The Android device will follow a similar procedure, except it will use LipeRMI to access the feedback code on the PC’s USB and the Android doesn’t use the metamorphosis code [11]. Figure 14 shows how the feedback code and metamorphosis code generate the new seed (Sx’).

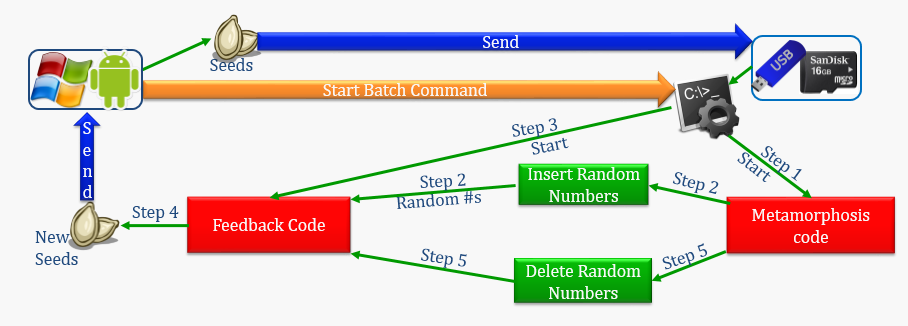


Figure 14. Feedback/Metamorphosis procedure

Java RMI is a Java library that allows remote method invocation and is used to allow the main program to invoke the methods in the feedback and metamorphosis codes that are stored on the USB [10]. LipeRMI is a lightweight version of Java RMI that is capable of running on the Android device (Java RMI isn’t supported by Android) [11]. Figure 15 shows how the feedback and metamorphosis codes are started by the batch file and then accessed from the PC and Android devices.

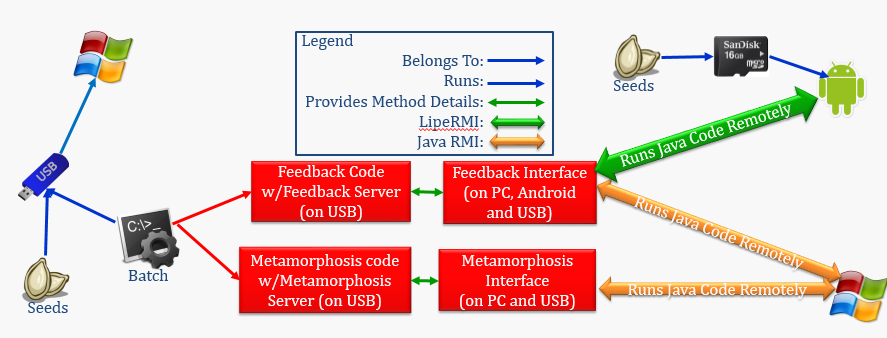


Figure 15. JavaRMI/LipeRMI and external storage codes

The USB device will contain the following items; Feedback.java, FeedbackInt.java, Metamorphosis.java, MetamorphosisInt.java, run\_Feedback.bat, Seed.txt, and Key.dat. The Android’s SD card will only hold Seed.txt and Key.dat. The FeedbackInt.java and MetamorphosisInt.java are java interfaces that are necessary for Java RMI and LipeRMI to work [12]. Seed.txt will hold the seed for generating the initial table, used to obfuscate the first three button presses. Key.dat will be the password encryption/decryption key; Key.dat will be encrypted using the seed that was generated by the feedback code (Sx’). Table 1 details the files that are stored on the USB drive and SD card.

|  |  |
| --- | --- |
| Files on USB | Files on SD |
| * Feedback.java - Program for generating the seed to create the master table and encryption/decryption key for Key.dat * FeedbackInt.java – Interface needed for Java RMI and LipeRMI remote method invocation libraries * Metamorphosis.java - Program for modifying the random number variable in Feedback.java * MetamorphosisInt.java - Interface needed for Java RMI and LipeRMI remote method invocation libraries * run\_Feedback.bat - The batch file for running Feedback.java and Metamorphosis.java * Seed.txt - The seed for generating the initial table * Key.dat - The encrypted key that is used to encrypt/decrypt the passwords for each communication channel | * Seed.txt - The seed for generating the initial table * Key.dat - The encrypted key that is used to encrypt/decrypt the passwords for each communication channel |

Table 1. Files stored on external storage

### 2.2.6 AES GCM Encryption

This design utilizes the AES-256 GCM encryption standard for encrypting any data in the system. AES (Advanced Encryption Standard) is a block cipher encryption scheme that was designed to replace DES (Data Encryption Standard) [13]. Figure 11 shows the encryption process for AES.

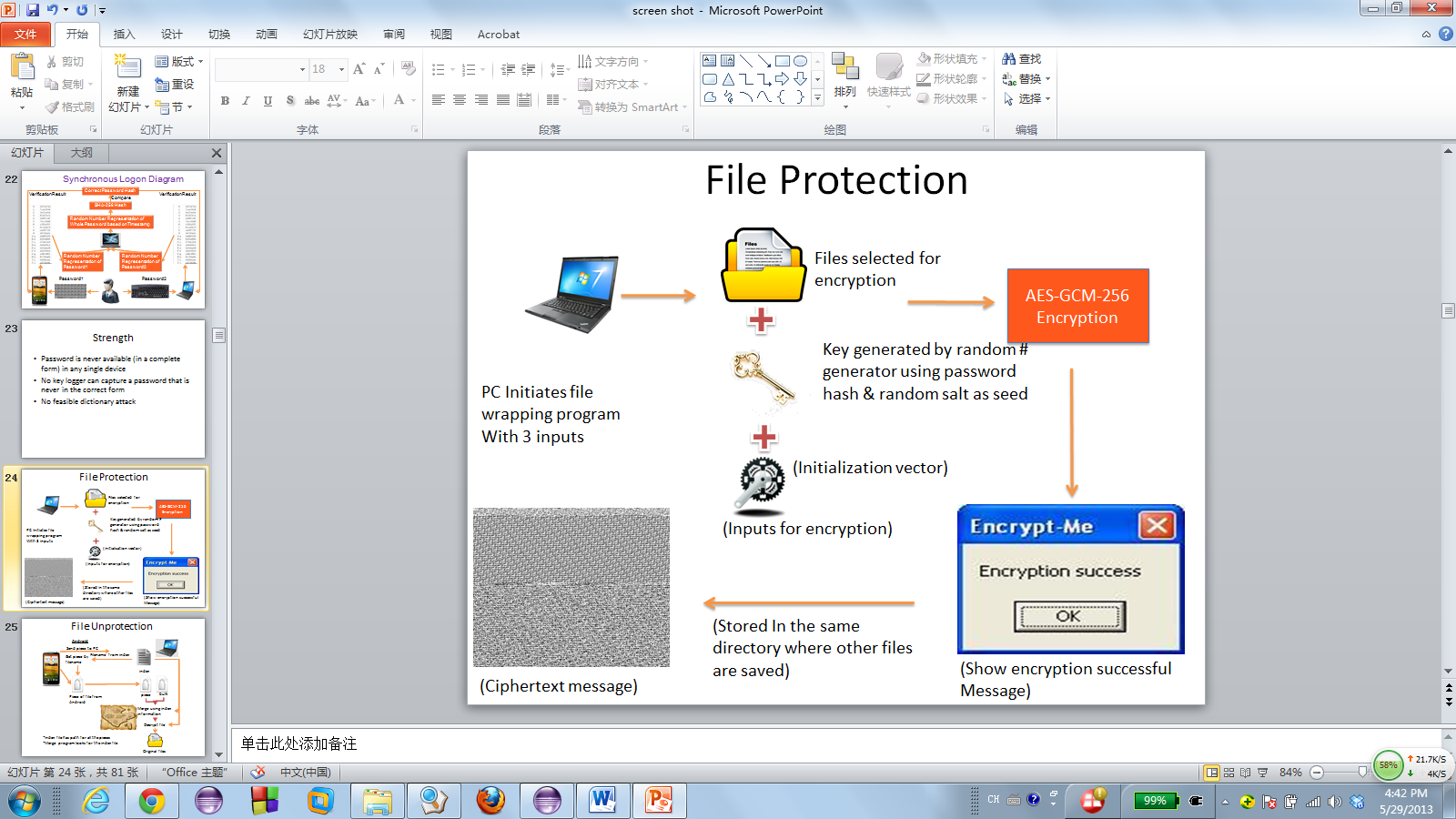


Figure 11. AES encryption process [14]

### 2.2.7 Document Splitting

This system utilizes page-wise document splitting to further defeat the attacker. The most vulnerable point of a protected document is when it resides in the memory post decryption. Page-wise document splitting only allows one page of any protected document to reside decrypted in the memory, thus minimizing the loss during a memory scraping attack to one page of a document. This system only supports PDF documents, with the PDFBox library, but file format support can be expanded by using additional Java libraries [1]. This component also adds each of the documents to a map file. Figure 12 shows how the directory is mapped and added to the map file and then each file in the directory enters the protection process.

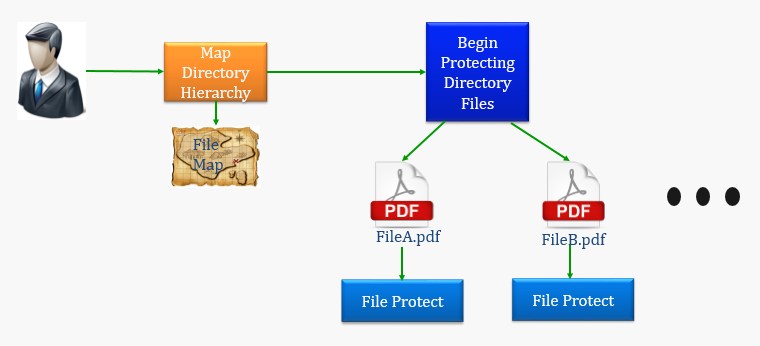


Figure 12. Protecting directory

Each document is split page-wise before it is protected. Each document has its filename, file path, and number of pages in the document added to the map file. The output of the splitting process are N PDF files where N is the number of pages of the original document. Each of these pages will then be encrypted, split, and distributed across the PC, Android, and server.

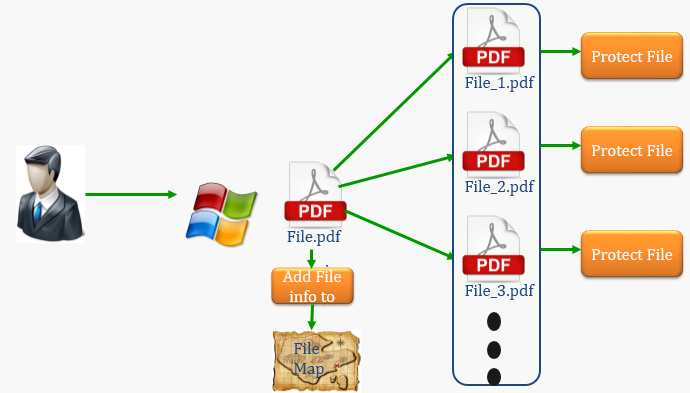


Figure 13. Document splitting

### 2.2.8 File Splitting and Merging

This system will use file splitting and distribution amongst the user’s devices to further defeat the attackers. Sun’s method of file splitting and merging is used for this portion of the design [14]. File splitting will be used on the files that are selected for protection and on the index file that details the locations of the protected files. Figure 14 shows the procedure for splitting the files. The encrypted file is split, where 1/5 of the contents are saved into the piece file and 4/5 of the contents with random numbers are saved into the bulk file. The piece file is saved on the Android and the bulk file is saved on the PC.

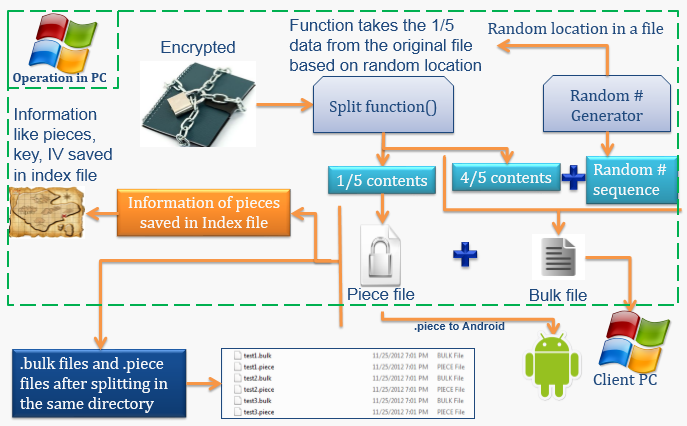


Figure 14. File splitting procedure [14]

During the file splitting procedure an index file is created to keep track of where all of the file pieces are located for the unprotection process [14]. The process for creating the index file is shown in Figure 15.

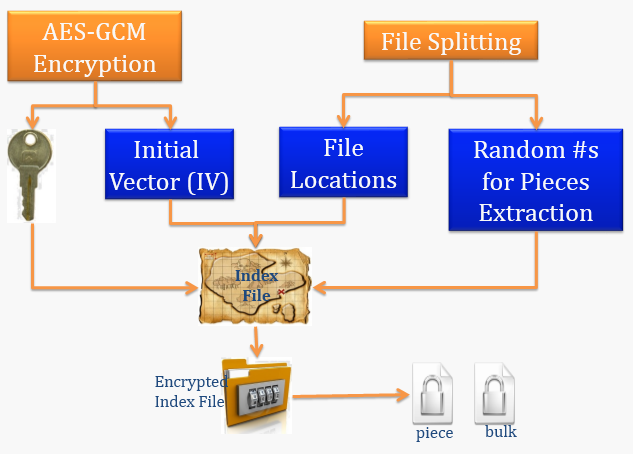
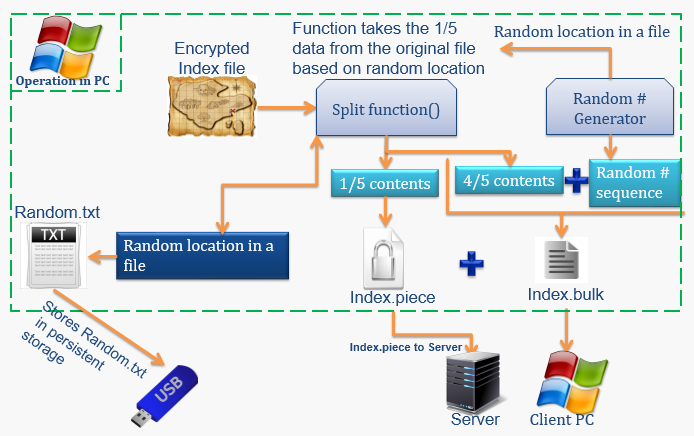


Figure 15. Index file generation [14]

Figure 16 shows the procedure for splitting the index files. This procedure is the same as the procedure for splitting encrypted files, except the piece file is saved on the server and instead of an index file there is a random.txt file.

Figure 16. Index file splitting procedure [14]

The split protected files and split index files will be distributed amongst the devices as in Figure 17 (also shows the overall procedure of encrypting/splitting the file and the index file. The server will save Index.piece, the Android saves File.piece, and the PC saves Index.bulk and File.bulk [14].

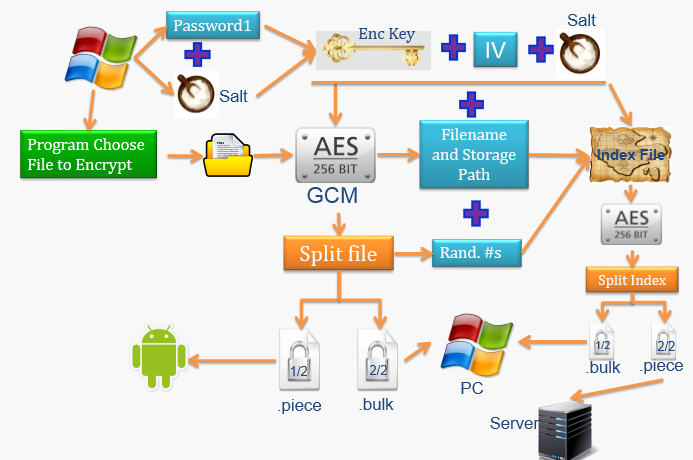


Figure 17. File/Index file encrypt and splitting procedure [14]

Figure 18 shows the procedure for merging the index files. The random.txt file is loaded in from the USB device and used to merge the piece and bulk files, after they are retrieved from the server and PC respectively. The output is the encrypted index file. Figure 19 shows the index merging procedure with the edition of decrypting the index file.

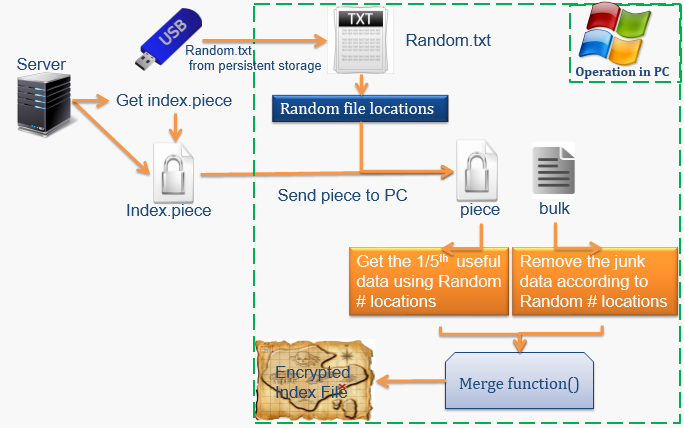


Figure 18. Index file merging procedure [14]

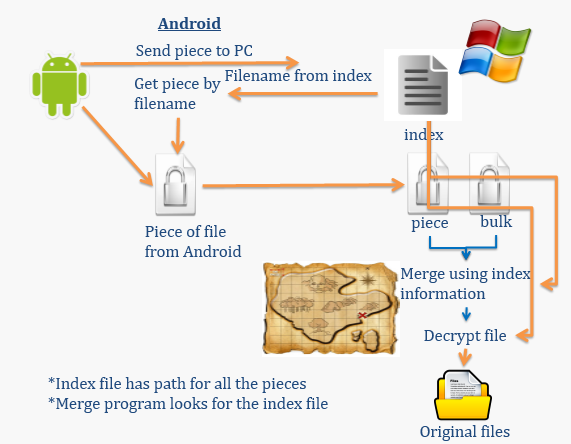


Figure 19. Index file merging and decryption procedure [14]

Figure 20 shows the procedure for merging the files. This procedure is the same as the procedure for merging the index files, except the piece file is retrieved from the Android and instead of a random.txt file there is an index file.

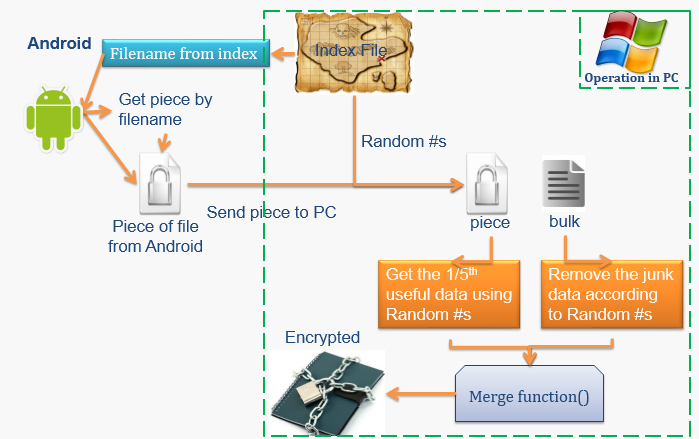


Figure 20. File merging procedure [14]

Figure 21 shows the procedure for merging the index files, using the decrypted index file to merge the data file, and then decrypt the data file to get the original file. The procedure for merging the file is just opposite of the splitting procedure.

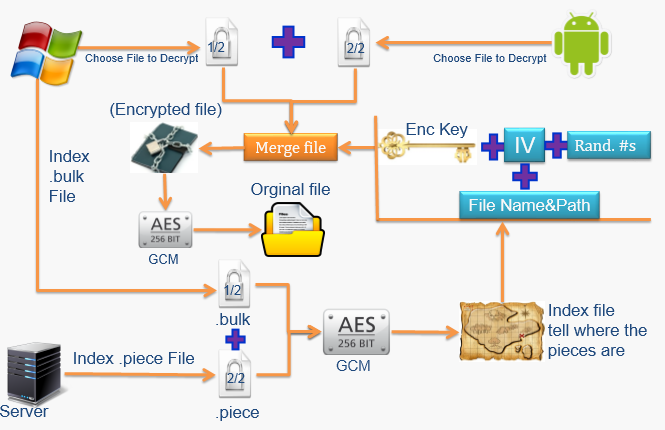


Figure 21. File merging and decryption procedure [14]

### 2.2.9 Protected File Browser

The protected file browser is used to display all of the files that have been protected and the number of pages each protected document has. This component also gives the user the ability to select a file and a page they want to view and open it. Figure 22 shows the protected file browser window.

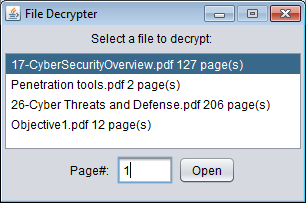


Figure 22. Protected file browser window

The overall procedure of browsing the protected files, unprotecting a page, displaying the page, and re-protecting the page are outlined in Figure 23. The protected file browser reads the map file that was created by the document splitting component and displays the contents. The user selects a document and page to unprotect. The piece and bulk files are merged and decrypted to get the PDF file for the selected page of the desired document. The document is then displayed in the default PDF viewer. When the user closes the PDF file it is then re-protected.

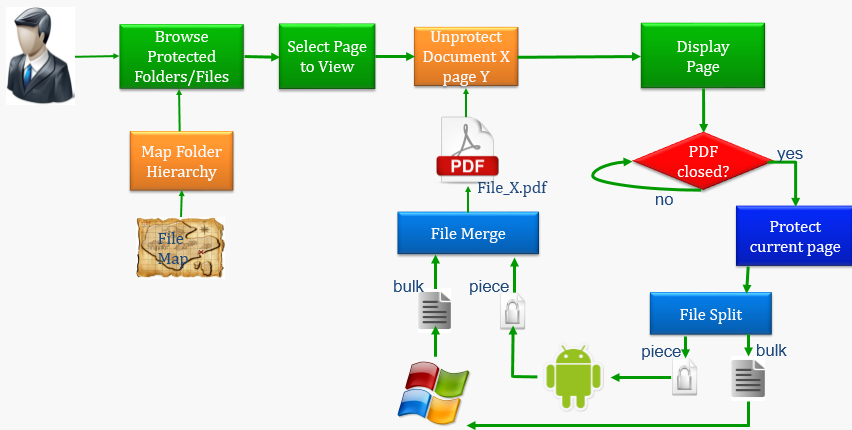


Figure 23. Document/page unprotect, display, and re-protect procedure

### 2.2.10 Software Keyboard

The software keyboard used for password input has been designed to offer superior protection against key-logging and screen capture attacks as compared to traditional keyboard input. Three primary aspects of the keyboard are responsible for this enhanced security: intra-session key scrambling, inter-session key value scrambling, and multi-valued inputs. To enhance user readability, the keys are color-coded according key values.

Intra-session key scrambling is carried out according to two separate queues: user input and dynamic time intervals. Any key pressed by the user triggers a scramble; additionally, scrambling occurs whenever a dynamic time interval between four and six seconds elapses. Figure 24 shows an example of key scrambling triggered by user input. Figure 24 also highlights the button layout – note that each button has four associated values. This ensures that the exact password of a user is never stored (even in obfuscated and encrypted form). Even if the obfuscation and encryption were to be cracked, the plaintext password would not be recovered.



Figure 24. Intra-session key scrambling.

For every login after registration, two keys have half of their values exchanged. The details of this process (i.e. which keys and what values) are derived from a generated seed. Figure 25 provides an example of how the key value swapping occurs.

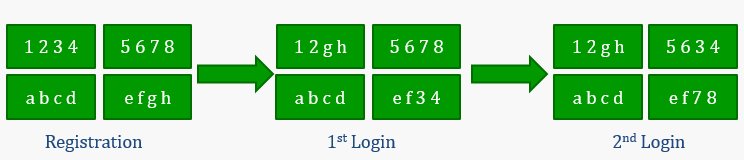


Figure 25. Inter-session key value swapping.

# 3. Demonstration

The system is very straightforward to use, although the PC software keyboard does take some time to get used to. The first step is to enter the server IP and client IP on the PC device, the window shown in Figure 26.

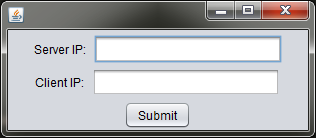


Figure 26. PC IP information input box

Next the server IP is to be input into the Android device as shown in Figure 27.

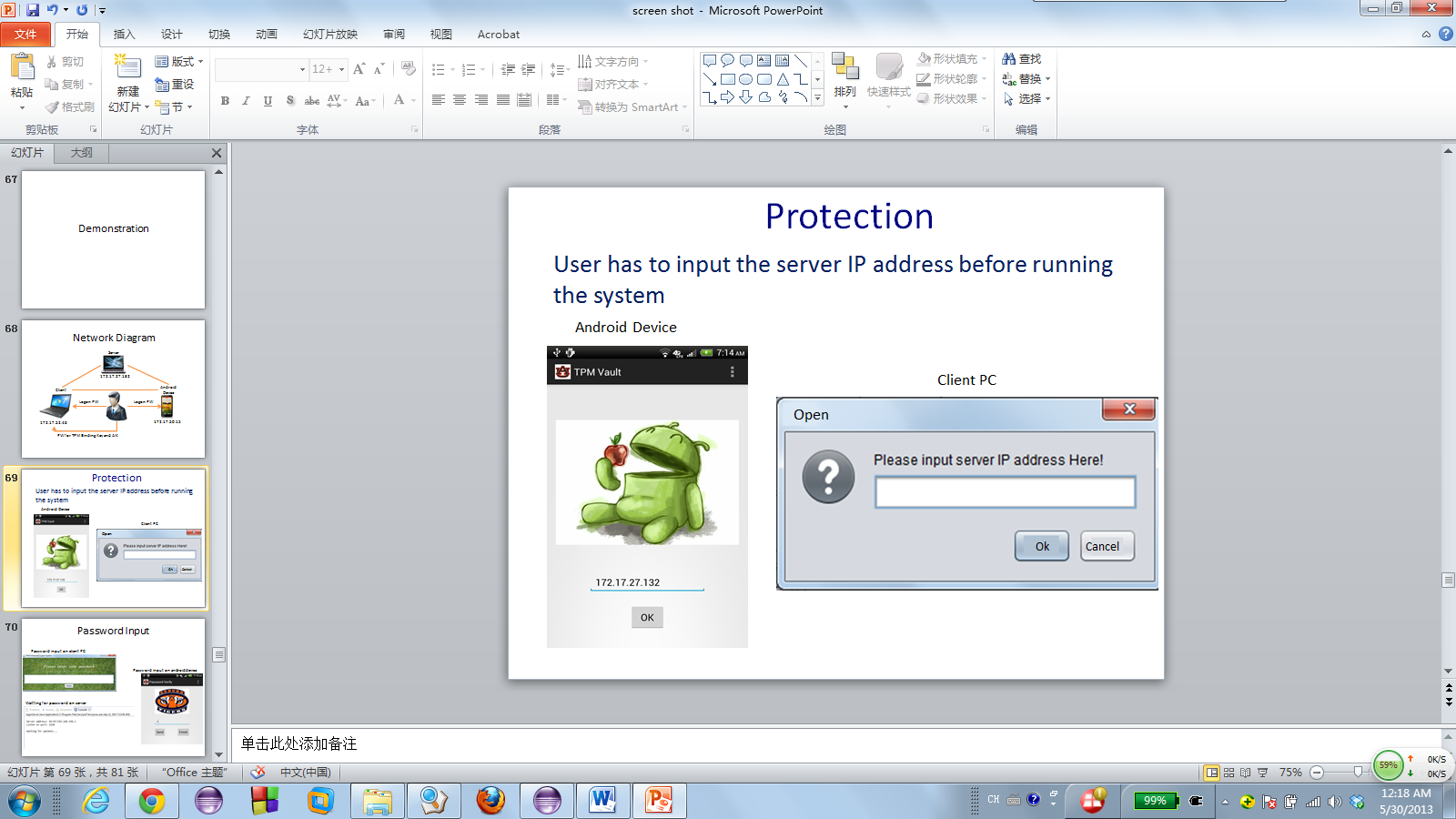


Figure 27. Android server IP input screen

Next the user is to enter their username (using their physical keyboard), created at registration, and then begin entering their password on the PC and Android devices (note the order in which the password is entered between the two devices matters). The PC portion(s) of the password are to be entered in the software keyboard, shown in Figure. Each button of the software keyboard represents four characters to enter the password just click on the button that contains the character you need to enter (note the password is case sensitive). Each character grouping keeps its color throughout the duration of the logon procedure and key randomization doesn’t start until after the fist key is clicked, allowing the user to get an idea of the colors associated with each key grouping.

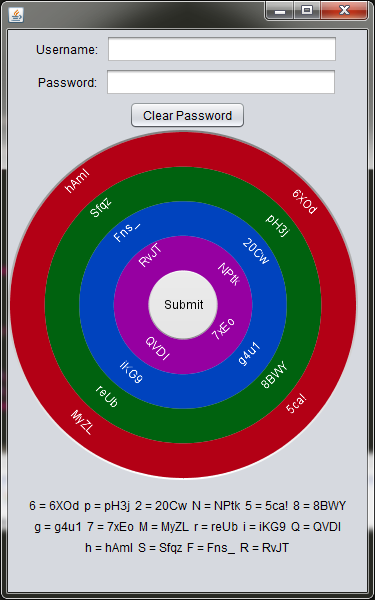


Figure 28. Software keyboard

The Android device uses the stock Android keyboard for password input, Figure 29. The user must look at the text below the PC software keyboard to translate the stock Android keyboard to key groupings (e.g. the key grouping 6XOd corresponds with the 6 key on the stock Android keyboard). After the password has been entered the user must click Finished on the Android and then Submit on the PC software keyboard.

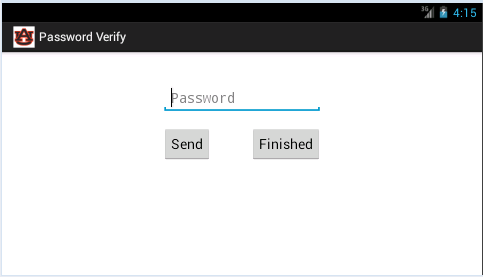


Figure 29. Android password entry screen

After a successful logon the user Protection/Unprotection window is displayed, shown in Figure 30.

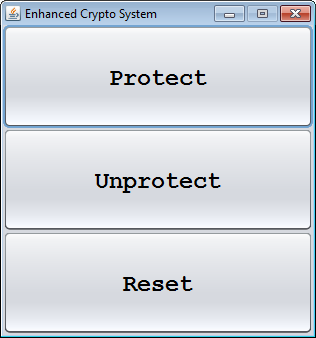


Figure 31. Protect/Unprotect window

Clicking the protection window opens up the directory/file selection window, shown in Figure 32. The user can select any combination of directories and files to protect; only PDF files will be protected all other file formats will be ignored. Clicking the open button will open the Protection progress window, shown in Figure 33, that shows the progress of the current job, the PDF file(s) being split, and the PDF files being protected (individual pages). When the protection procedure is done a protection finished message will be displayed at the bottom as shown in Figure 34.

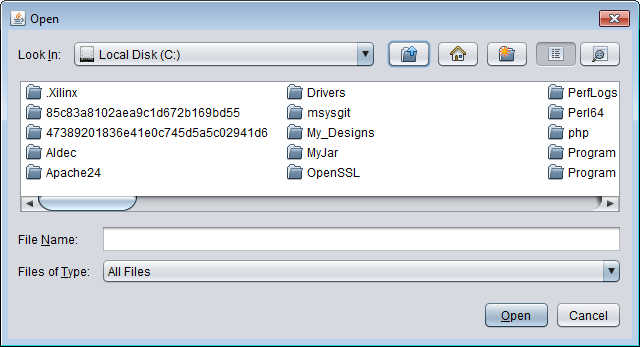


Figure 32. Choose directories and or documents to protect

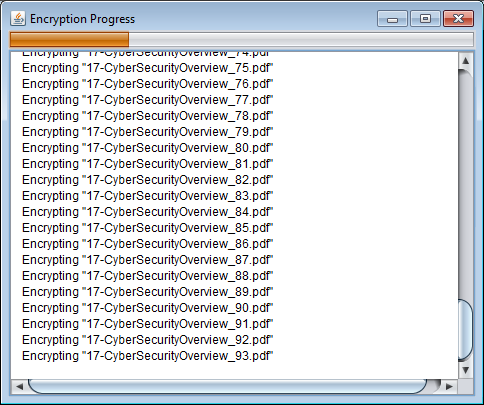


Figure 33. Directory protection in progress

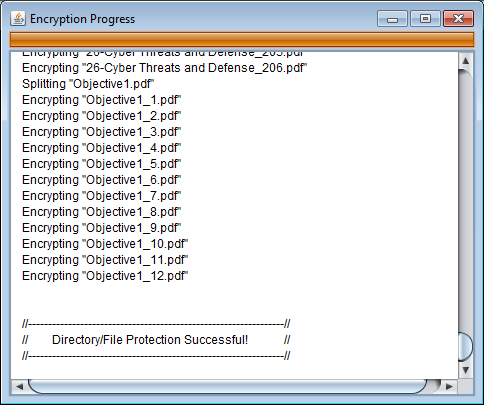


Figure 34. Directory protection after successful completion

When the user clicks unprotect on the Protection/Unprotection window the file unprotection window will be displayed, shown in Figure 35. The user just selects the file they want to view and enters in the page number of that file they want to view and clicks the open button. When the user clicks open the page of the document they want to view is opened in their default PDF viewer, as shown in Figure 36. When the user is finished viewing the opened page of the document they must first close the PDF page and then click the ok button in the dialog window shown in Figure 37 causing the page to be re-protected. If the user clicks ok in the dialog box before closing the PDF document the PDF document will not be re-protected.

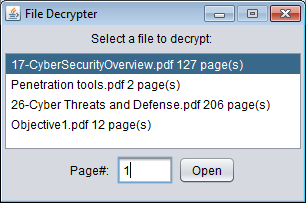


Figure 35. File Protection window

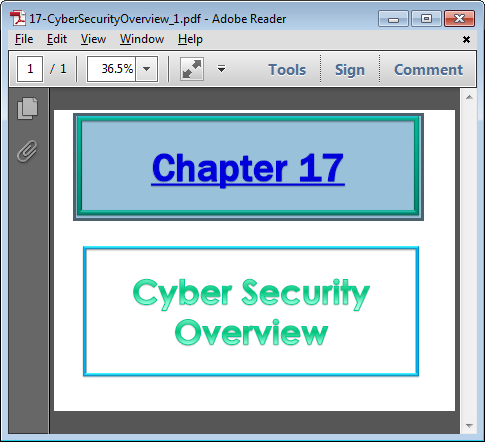


Figure 36. PDF page opened in default PDF viewer

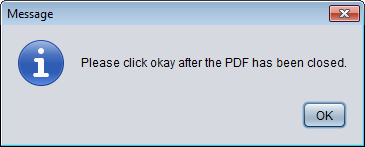


Figure 37. Click ok after PDF document has been closed

# 4. Results

## 4.1 Performance

Several performance metrics were taken; logon latency (seconds), PDF split/protect latency (seconds), PDF split latency (seconds), protection latency (milliseconds), unprotection latency (milliseconds). The logon latency is shown in Table 2 and includes time to enter password + network latency + authentication latency. The logon latency is anywhere between 25 to 35 seconds, although this number would be higher for someone just beginning on this system, and the average is about 29 seconds.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average |
| Logon Time | 29.1 | 27.1 | 38.6 | 30.1 | 26.4 | 30.8 | 26.0 | 21.5 | 34.4 | 23.8 | 28.78 |

Table 2. Logon latency (user input time + network latency + authentication latency) (seconds)

The next metric taken was the PDF split/protect latency, measured in seconds, which was the PDF split latency + file protect latency. The PDF split/protect latency data is shown in Table 3. This data shows the latency increases as file size and number of pages increase. The PDF split/protect latency can be time consuming for large documents (or directories), but it is a one-time process, so the documents only need to be split and protected once.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group  File size  & number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average |
| File number: 1  File size: 50KB  # of Pages: 6 | 0.27 | 0.14 | 0.11 | 0.11 | 0.11 | 0.11 | 0.12 | 0.10 | 0.11 | 0.11 | 0.129 |
| File number: 2  File size: 1100KB  # of Pages: 26 | 1.84 | 3.21 | 2.32 | 2.64 | 1.82 | 1.90 | 2.27 | 2.06 | 2.93 | 2.15 | 2.314 |
| File number: 3  File size: 2000KB  # of Pages: 103 | 21.6 | 20.3 | 22.5 | 19.0 | 20.8 | 20.5 | 22.9 | 21.5 | 19.5 | 19.7 | 20.83 |
| File number: 5  File size: 3000KB  # of Pages: 147 | 35.1 | 31.3 | 30.2 | 33.1 | 31.9 | 32.8 | 32.7 | 36.1 | 32.5 | 31.6 | 32.73 |
| File number: 2  File size: 5600KB  # of Pages: 170 | 39.2 | 37.3 | 36.1 | 35.8 | 39.9 | 35.4 | 39.0 | 37.5 | 34.7 | 38.5 | 37.34 |

Table 3. PDF Split/Protect latency (PDF split latency + protect latency) (seconds)

The next performance metric taken was the PDF split latency alone (measured in seconds). This data is shown in Table 4. This data shows the latency increases as file size and number of pages increases. The PDF split can be time consuming for large documents but it is a one-time process.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group   File size  & number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average |
| File number: 1  File size: 50KB  # of Pages: 6 | 0.07 | 0.08 | 0.07 | 0.07 | 0.07 | 0.08 | 0.08 | 0.06 | 0.07 | 0.08 | 0.073 |
| File number: 2  File size: 1100KB  # of Pages: 26 | 1.18 | 1.78 | 1.12 | 1.34 | 1.15 | 1.25 | 1.45 | 1.16 | 1.84 | 1.23 | 1.350 |
| File number: 3  File size: 2000KB  # of Pages: 103 | 9.49 | 8.76 | 9.70 | 9.20 | 9.17 | 9.82 | 7.90 | 10.1 | 9.35 | 8.85 | 9.230 |
| File number: 5  File size: 3000KB  # of Pages: 147 | 15.6 | 14.2 | 15.8 | 15.1 | 15.1 | 16.3 | 16.2 | 16.7 | 16.5 | 15.5 | 15.70 |
| File number: 2  File size: 5600KB  # of Pages: 170 | 18.2 | 17.8 | 18.1 | 15.0 | 21.3 | 18.5 | 18.2 | 17.1 | 17.2 | 17.7 | 17.91 |

Table 4. PDF split latency (seconds)

The next performance metric taken was the file protect latency alone (measured in milliseconds). This data is shown in Table 5. This data shows the latency increases as the file size increases. Since a single page PDF file doesn’t very much in size the times won’t get much larger than about 30 milliseconds, which will be instantaneous from a user’s perspective.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group  File size  &number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average |
| File number: 1  File size: 50KB | 15.0 | 12.3 | 11.5 | 7.78 | 12.5 | 9.90 | 9.33 | 7.78 | 7.57 | 8.92 | 10.258 |
| File number: 2  File size: 100KB | 18.7 | 11.8 | 15.6 | 14.4 | 14.3 | 10.5 | 10.5 | 13.7 | 9.23 | 12.6 | 11.703 |
| File number: 3  File size: 200KB | 16.5 | 14.7 | 14.4 | 20.7 | 18.8 | 20.6 | 24.7 | 16.2 | 16.8 | 15.4 | 17.880 |
| File number: 5  File size: 300KB | 29.0 | 25.1 | 23.2 | 22.4 | 25.2 | 17.9 | 20.0 | 21.4 | 28.1 | 17.6 | 22.990 |
| File number: 2  File size: 500KB | 47.1 | 24.3 | 25.5 | 37.3 | 26 | 36.7 | 29.7 | 25.4 | 26.5 | 23.6 | 30.210 |

Table 5. Protection latency (milliseconds)

The last performance metric taken was the file unprotect latency alone (measured in milliseconds). This data is shown in Table 6. This data shows the latency increases as the file size increases. Since a single page PDF file doesn’t very much in size the times won’t get much larger than about 100 milliseconds, which will be instantaneous from a user’s perspective.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group  File size  &number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average |
| File number: 1  File size: 50KB | 19.2 | 28.0 | 37.9 | 29.2 | 46.7 | 15.0 | 33.6 | 18.8 | 23.6 | 15.4 | 26.74 |
| File number: 2  File size: 100KB | 48.8 | 19.0 | 18.2 | 40.0 | 30.8 | 35.1 | 18.7 | 35.3 | 44.9 | 20.3 | 31.11 |
| File number: 3  File size: 200KB | 44.2 | 25.9 | 46.8 | 27.9 | 44.7 | 26.9 | 25.1 | 43.6 | 26.5 | 28.3 | 33.99 |
| File number: 5  File size: 300KB | 61.2 | 38.9 | 30.2 | 34.4 | 37.7 | 54.4 | 34.5 | 42.7 | 48.6 | 35.4 | 41.8 |
| File number: 2  File size: 500KB | 95.3 | 60.7 | 61.3 | 54.5 | 66.6 | 76.5 | 77.0 | 106 | 48.1 | 70.4 | 71.64 |

Table 6. Unprotection latency (milliseconds)

# 6. Security Analysis

Developing a secure system requires careful consideration of the attack vectors that may be used to potentially compromise it. The following section will discuss in detail how each module utilized in this system defeats one or more attack vectors that would otherwise be considered lethal. Table 7 provides a summary in brief to that effect.

|  |  |
| --- | --- |
| Attack Method | Defense |
| * Software/Hardware Keyloggers * Screen Capture, HTML injection | * Multi-device, distributed password input * Random character groupings * Intra-session key scrambling * Inter-session key value scrambling |
| * Password Guessing (Brute force) * Dictionary Attack * Device Spoofing | * Pre-negotiated information requirement * System suicide procedure |
| * Packet Sniffing * Middle man attack | * GCM-AES-256 authenticated encryption for transmission channels between all devices without PKI |
| * Physical theft of assets * loss of all user’s devices | * Cloud-based data backup * User’s knowledge cannot be captured using any technical method |
| * Malware | * Distributed file protection scheme |
| * Spear Phishing | * Input device password obfuscation |
| * In-time memory scraping | * Page-wise file access |

The authentication/authorization system implemented in this design was created with the intent of defeating several attack vectors considered lethal to other systems. By forcing the use of a software keyboard for password input, hardware keyloggers are negated. Since each button on the software keyboard represents a grouping of characters (rather than a 1:1 mapping of button to character), a user’s true intended password cannot be gleaned from a software keylogger. Additionally, each character in the grouping is mapped to a random 8-character string according to a mapping table generated dynamically at login. The table is itself modified during login based on initial user input, further strengthening the obfuscation. Intra-session key location scrambling occurs based on both user input and a random time interval between four and six seconds, providing sufficient randomness to render screen capture attacks useless.

Successful authentication requires specific information generated during the previous login (or registration) and stored on an external medium. Without this information, which is backed up using cloud-based storage, a successfully verifiable password cannot be generated. This feature inherently prevents the spoofing of a client device. The inclusion of cloud-based backups provides enhanced convenience and accessibility for an authorized user in the face of loss/destruction of physical assets.

A built-in suicide procedure exists within the system that will destroy the previously mentioned authentication information stored externally. This procedure is executed after a fifth sequential failed logon is attempted. Password brute-forcing, dictionary attacks, and password guessing are all handedly defeated by this measure – even with the actual text of the password, attackers must still utilize the correct input order across devices.

Packet sniffing attacks, while not overly difficult to defeat, can still pose a threat to the security of a system if ignored. This system utilizes communication channels between devices secured with GCM-AES-256 encryption. The encryption/decryption key for each channel is derived at logon by each pair of devices.

Many systems are vulnerable to single points of failure, and it is often the case that malware provides attackers with the means to exploit this vulnerability. Encrypted files, if stored in a complete form on one device, can be stolen by an attacker and eventually decrypted. The distributed nature of the file protection system utilized in this design ensures that every device in the system must be similarly compromised in order to reassemble any particular file. File bulk, piece, and index files are separately encrypted and stored across multiple devices, and each item must be successfully decrypted before assembly can begin.

Perhaps the most lethal attack vector currently known is in-time memory scraping. By waiting until an authorized user bypasses security measures and accesses sensitive information, an attacker can defeat even the most sophisticated systems. Data stored in memory (in plain text) is vulnerable, and very little can be done to erase this vulnerability. While this system does not provide perfect protection against memory-based attacks, it does take measures to limit the risk of data leakage. All documents are split page-wise before any protection takes place, and documents are accessed a single page at a time; furthermore, the accessed pages are immediately re-protected after the user has finished reading it. This limits both the window during which the information is vulnerable and the quantity of information that could possibly be lost.

As with almost any security system, the human element is the weakest aspect. Complete system compromise could be achieved by tricking or coercing the user into divulging the complete details of his/her password in addition to the credentials necessary to access the cloud backup services. This risk could be mitigated by ensuring that a different individual controls the cloud services; however, the user could still relinquish the physical assets (i.e. external storage devices) and, in conjunction with revealing his/her password, compromise the system.

# 7. Conclusion

Our system provides reliable file security in the face of unknown attack vectors, including limited protection against in-time memory scraping attacks. Though the obfuscated authentication/authorization procedure may at first seem onerous to a user, measures were taken to ensure that a user can quickly adapt to the input method. Use of cloud-based storage provides users with reliable accessibility to their protected data even in the event of loss/destruction of physical assets. In contrast, unauthorized users attempting to access protected data through brute-force or side-channel attack methods will find their efforts in vain thanks to the built-in suicide procedure. The page-wise protection measures applied to protected files require a reasonable overhead in the order of 100 ms.

# 8. Works Cited

[1] The Apache Software Foundation, “Apache PDFBox - A Java PDF Library,” *pdfbox.apache.org*, 2014. [Online]. Available: http://pdfbox.apache.org/.

[2] National Institute of Standards and Technology, *{FIPS PUB 197}: Advanced Encryption Standard (AES)*. NIST, 2001.

[3] M. Dworkin, *{NIST 800-38D}: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST, 2007.

[4] J. Postel, *{RFC 768}: User Datagram Protocol*. IETF, 1980.

[5] “TCP vs UDP - Difference and Comparison | Diffen.” [Online]. Available: http://www.diffen.com/difference/TCP\_vs\_UDP. [Accessed: 19-Jan-2014].

[6] “Advantages and disadvantages of Stream versus Block Ciphers.” [Online]. Available: http://security.stackexchange.com/questions/334/advantages-and-disadvantages-of-stream-versus-block-ciphers. [Accessed: 02-Feb-2014].

[7] “Java Core API,” *Dropbox*. [Online]. Available: https://www.dropbox.com/developers/core/start/java. [Accessed: 01-May-2014].

[8] National Institute of Standards and Technology, *{FIPS PUB 180-2}: Secure Hash Standard*. NIST, 2002.

[9] C. Kaufman, R. Perlman, and M. Speciner, *Network Security Private Communication in a Public World*. Prentice Hall PTR, 1995.

[10] Oracle, “Remote Method Invocation Home,” *oracle.com*. [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[11] F. Andrade, “lipe rmi a light weight internet approach for remote method invocation,” *lipermi.sourceforge.net*, 2006. [Online]. Available: http://lipermi.sourceforge.net/.

[12] M. Hall and L. M. Brown, “A Very Simple RMI Example: The Four Required Classes,” *apl.jhu.edu*, 1999-1996. [Online]. Available: http://apl.jhu.edu/Classes/Notes/LMBrown/courses/605-481/notes/Client-Server/RMI-Simple-Example.html.

[13] C.-H. (John) Wu and J. D. Irwin, *Introduction to Computer Networks and Cybersecurity*. CRC Press, 2013.

[14] S. Hao, “MEE Final Report-5-30-HAO SUN.” .

# Appendix

* Before running this program be sure to create a test directory and make a backup of the files that are put in this directory. This program does not support any method for reassembling the original document after it has been split and protected (i.e. if the original PDF was 100 pages it will remain 100 separate PDF files). \*\*If the C drive is selected as the directory to protect this program will split and protect every PDF file on your hard drive and it will take a long time to manually rebuild each file\*\*
  + If a PDF document accidently selected for protection one way to reassemble the document is to unprotect each page individually and click the ok button before closing the PDF (this keeps the PDF from being re-protected) and then using the PDFBox API to rebuild the documents programmatically (using merge).