BEPIS GOLA SECRET RECIPE VAULT

GROUP REPORT FOR INTRODUCTION TO C PROGRAMMING- 48430, AUT2018

Ben Kelly - 12544514
Jack Christie - 13216039
Tom Pugh - 12562385
Rachel Birrell - 13206622
Leah Schirmer - 13197761

Table of Contents

Introduction

The Bepis Gola Secret Recipe Vault (SRV) is a standalone software package which stores a secret recipe in a secure, yet accessible form. This system aims to solve the problem of storing data securely, using methods of encryption and compression. Although it is a very niche context, the application of such a system is quite practical for a medium business when it comes to storing classified information, as well as many other industries that require the use of encryption in their storage. Moreover, we have used a polynomial key system which allows for more flexible access to the recipe, whilst still maintaining security to a high standard. We the developers believe it is a perfect solution to the problems Bepis are facing.

1 About the Client

Our client is the manager of a large, multinational (and yes, fictitious) drinks corporation called Bepis Gola (spin on the name “Pepsi Cola”). The company has been producing beverages for decades and hence, have a secret formula for their drink. This secret recipe is in bitmap format (BMP file format). Up until now, their five board members (executives) have had no digital security measures in place to thwart potential thieves- until a recent incident involving a malicious employee convinced them to approach us for a new system of secure recipe storage.

2 Problem Definition and Analysis

Ultimately, we have been tasked with finding a way to securely protecting the secret recipe from any outsider threats. To do this we need to set up a secure system which may use encryption as a method of protecting it. Due to the fact that it will reside on the same system as other servers (eg. sales) the storage system needs to take up as little space as possible to avoid taking up too much space.

The client has stated they want the utmost security as to prevent further attacks and that they need to be able to easily maintain and correct it when things go wrong. They have also outlined a particular operation mode, whereby each board member gets a personal key upon storing, but only three are required to retrieve the recipe. That way if members are absent the recipe can still be retrieved. The board wants the solution completed and working as soon as possible.

2.1 Client Requirements

- The system must store their recipe in an encrypted format that can only be decrypted by the executives.
- For added security the client would like at least three executives to be present when viewing or accessing the recipe, instead of five.
- The program needs to have a debugging mode, to avoid any unnecessary bugs or faults that could be present.
- The time frame to implement the solution is 1 month.
- The file systems need to take up as little space as possible to ensure that there is enough space other company files.

2.2 Scope of Solution

This will be designed as a custom solution for Bepis Gola and hence, we are only considering it in terms of the company, with particular focus on its only users; the board members. However, there is the potential for this system to be adapted to suit the needs of other industries, where compression and encryption are just as necessary in order to function. It is for this reason that we will attempt to make the source code clearer in terms of intrinsic documentation and debugging, so that it may be easily modified if such a time arises.

3 Solution Options

3.1 Option 1 – Full, Image Based Encryption with Added Compression

Option one involves creating a secure system to protect the file, which uses both compression and encryption to safely store the file. Each of the five board members would be given a "key", which is a set of two dimensional coordinates when the file in added to the database. These keys would be needed by at least three of the executives in order to view the recipe. Due to the recipe files being quite large, the image will need to be compressed before being encrypted and stored.

3.2 Option 2 – Basic, Text Based Encryption with One- Key Access

Option two involves creating a system that converts the recipe from BMP file format to a text file format before encrypting it and storing it safely in the company database. During the encryption process each executive will be given a 'key' which consists of a two-digit number. This two-digit number would need to be used to view the recipe. However, due to the 'key' being a two-digit number only one executive is needed to access the recipe.

3.3 Option 3 – No Encryption

Option three takes the recipe in BMP file format and compresses it into a smaller file before storing it in the company database. This method relies heavily on the computer's security rather than the actual encryption method however, it means that the file will take up less space in the database. Each executive would be given a key that allows them to access the file. The keys would all be individual with no measures in place preventing one executive accessing the file on their own.

4 Selected Design

We have chosen to go ahead with option number one, due to the fact that it is the only one out of the three that correctly achieves all design requirements outlined in the previous section.

4.1 Program Features – From the Developer's Perspective

In terms of compression, we have decided to go with Huffman encryption due to its simplicity and effectiveness. It is optimised for bitmapped images, more specifically, colour photographs such as the recipe. Most importantly, it is lossless compression and hence no resolution or data is lost in the process of storing or retrieving a recipe. As part of Huffman compression, there is a bubble sort included, as well as a string search and replace function.

For encryption, we initially decided to use a division- modulus algorithm. However, there were severe issues to do with temporary data storage during the process. Instead, we experimented with using an exponential function, then a linear polynomial and finally we settled on using a simple XOR process as the heart of our encryption algorithm.

On top of this, to meet the requirement of only using 3/5 keys to retrieve, we are using a quadratic equation model. More information on this is provided in Key Generation.

For file storage, we decided to use images since they are easy to manipulate, considering their 2D nature. More specifically, we are using .bmp images since they are uncompressed, as opposed to other formats like .jpg.

4.2 Program Features and Flow – From the User's Perspective

The program has an interface that is simplistic and easy to use, ensuring that the user can save time by avoiding inconvenient unnecessary additions. Some of the features of the program are; an interactive command line menu, a standard menu inside the program and a debugging mode which are all coloured and visually appealing to the user.

The program starts with the command line, the user can either choose to input '-a' on the end of the command to add a recipe or '-v' to view the recipe. This will take the user straight into the program, skipping the internal menu. If the user chooses instead to run the program without entering arguments into the command line, then a separate menu will appear on the screen prompting the user to either enter '1' to add a recipe or '2' to view the recipe. From here, the program runs the user through the subsequent steps depending on the input however it follows the same path regardless of the commencing input methods taken.

Menu vs Command Line

The executable itself can be run in two ways:

- Command Line Method- commands and data are passed as arguments in the terminal, to the executable as it is run (-v for retrieve/ view recipe, -a for add and -h for help. More info on the correct syntax can be found in the help menu)
- Menu Method- if there were no arguments passed, the executable will present a numbered menu, where the user goes through a number of dialogues to select the desired action and enter the required data.

Debug Mode

The Software package also has a debugging mode, as per the client requirements. This will make it easier for the company to maintain and update the system as they please. The primary manifestation of this function is a verbose output to the terminal during operation, detailing exactly what is running and a small (but not to detailed) insight into the data being processed. It also includes more details in error messages.

4.3 How the System Works

The system has two primary modes of operation; add recipe and retrieve recipe. Each contain the same processes, but in reverse. Adding a recipe happens as follows:

- User points program to recipe file
- File is first compressed to take up as little space as possible
- File is then encrypted, where a randomly generated key (known as the master key) is produced
- The master key is split up using a polynomial equation process to generate five individual keys, in such a way that only three are needed for the reverse process.
- The encrypted file is then stored, and the five keys are given to each respective user

The Process for Retrieving a Recipe is as follows:

- User Points program to encrypted recipe file
- User inputs at least three individual keys
- Master key is derived based on the three keys
- File is decrypted based on master key
- Decrypted file is then decompressed, result is the original recipe
- Recipe is then returned to user

Key Generation

To generate multiple individual keys for each executive, we are going to use five points of a quadratic and use them to find the coefficients of a cartesian curve, that when combined will form the master key. Simply put, the quadratic equation (Ax2 + Bx + C) needs any 3 points to find the coefficients. In this case, the coefficients (A, B and C) form the master key for encryption, and the randomly chosen three/ five points (x, y) form the individual keys. This way, the master key can be derived from any three individual keys, and we can generate as many individual keys as we want, as more board members are hired.

4.4 Design Tools and Documentation

This section details the solution in terms of its real- life implementation. We will use a number of standard system design tools to convey the product itself. **Full size versions of diagrams in this section can be found in the appendices**, so these photos should not be referenced.

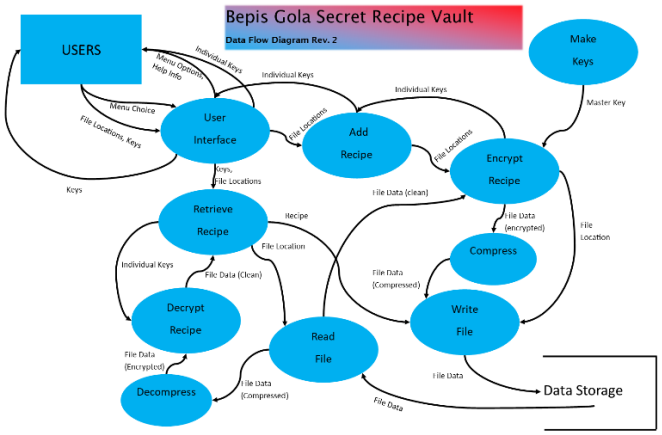

Figure 4.1: Data Flow Diagram (Original by Jack Christie)

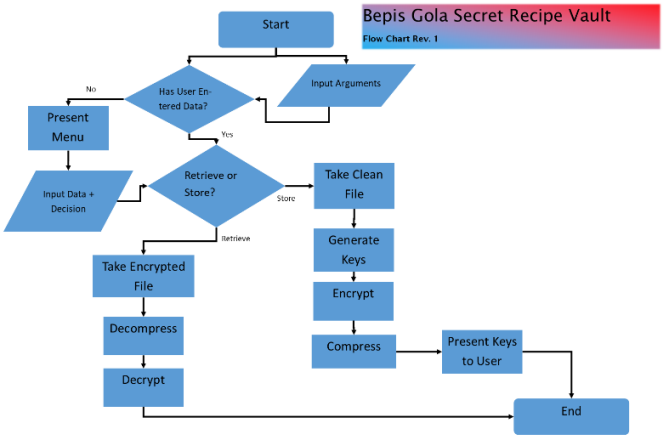

Figure 4.2: Flowchart (Original Graphic by Jack Christie)

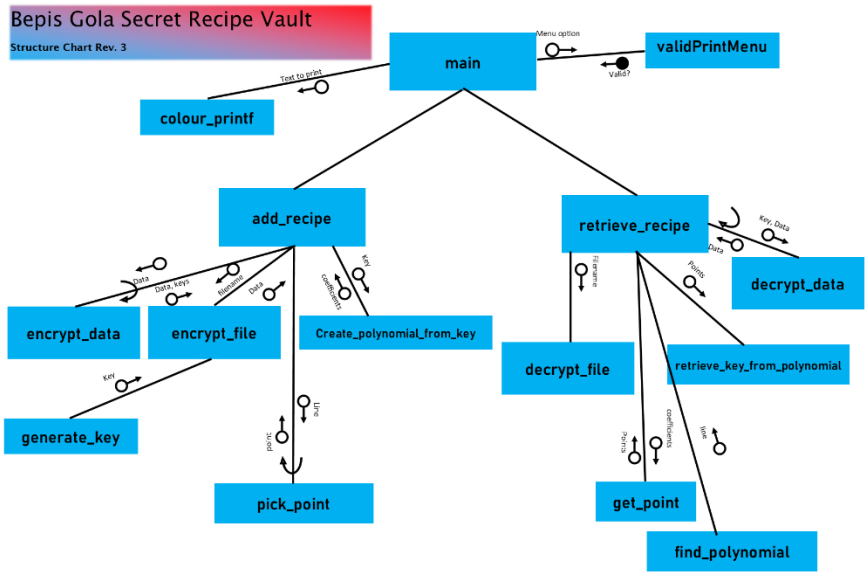

Figure 4.3: Structure Chart (Original Graphic by Jack Christie)

Table 4.1: IPO Chart

Function Name	Inputs	Processing	Outputs
retrieve_recipe	The file directory, three individual keys.	Retrieves the recipe from file storage, using the given keys.	int (bool) indicating if success
add_recipe	Filename.	Adds the recipe to file storage, returns keys to user.	int (bool) indicating if success
validPrintMenu	The choice entered.	Checks if the menu choice inputted was valid.	int (bool) indicating it is a valid menu choice
colour_printf	The desired colour, the text that you would like to be printed in colour.	Prints text in a specifically selected colour.	none
generate_key	None	Generates a random key, based upon the size of the file.	Generated key
encrypt_data	Key, unencrypted data & length of data	Encrypts a string with a key using XOR	Pointer to cyphertext
decrypt_data	Key, encrypted data & length of data.	Decrypts a string with a key using XOR	Pointer to plain text
encrypt_file	Key & file to encrypt	Encrypts a file by retrieving the clean file	int (bool) indicating if success, also outputs encrypted file into storage
decrypt_file	Key & file to decrypt	Decrypts a file	int (bool) indicating if success, also outputs decrypted file.
create_polynomial_from_key	Key to split into coefficients	Creates coefficients of a polynomial from a key	A polynomial object
pick_point	Polynomial	Picks a point in the 2D plane that lies on the parabola based on coefficients	A new point which lies on the polynomial

find_polynomial	At least 3 points that lie on the polynomial	Finds the coefficients of a polynomial based on the coordinates of 3 points.	A polynomial created from the 3 points.
retrieve_key_from_polynomial	Polynomial object	Retrieves a key from a set of coefficients of a polynomial.	Coefficients of p joined together as a key
char_to_buffer	A character to make into a buffer	Creates a new bit buffer from a character	Bit buffer, with the bits representing the character
buffer_to_char	Bit buffer	Converts the buffer back into a char	character
clear_buffer	Bit buffer (pointer)	Clears a bit buffer to all zeroes	Bit buffer (pointer)
display_buffer	Bit buffer	Displays a buffer, in its entirety, to the terminal	none
get_next_bit	Bit buffer (pointer)	Gets the next bit in a buffer.	Next bit in buffer. If no more, return -1
add_bit_char	Bit buffer (pointer), value to add	Adds a bit to a bit buffer.	New bit buffer (pointer), returns 1 in buffer is full
compress_file	The huffman code file, target file (pointer)	Compresses a file	The compressed file (pointer)
decompress_file	The huffman code, target file (pointer)	Decompresses a file	Decompressed file (pointer)
replace_in_string	String to replace text in, string to find, replacement string	Replaces embedded string (within a string) with another string	Replaced string (pointer)

5 Implementation and Testing

5.1 Implementation Plan and Data Migration Plan

We considered a few strategies for implementing the system, but we eventually settled upon a binary approach, whereby we will set up the new, computerised system at the same time the old system (a simple BMP file) will be decommissioned. In terms of data migration, the system will first be put to use when we move the recipe over from the hard drive it resides on, into a digital file, and then into the new system. This will be done by the executives themselves, with potentially some help from the development team.

5.2 Test Log

Table 5.2: Test Log

Date	What Was Tested (Include Process)	Results	File(s) Used
21/05/18	This was testing the code to open an image file, as well as a new text file and write the image file contents into the new text file.	The code was not able to write the binary of the image file, as it was writing the pointer location of the image file rather than the file itself.	PIC.bmp NewFile
22/05/18	Similar to the test done the day before, but with a few changes that would hopefully fix the process and write the binary of the image rather than the pointer location.	It was successful in writing the image binary into a new file.	PIC.bmp NewFile
22/05/18	Now that we had the Image => Text working well, this test was to see if the same code could be revered to convert that text back to its original text file.	Yes, it was able to be used inversely.	NewFile NewImage.bmp
23/05/18	Test Compress, we compressed an image and checked the file size before and after to see if it had actually compressed.	It was able to compress, indicated by the change in file size before and after compression.	PIC.bmp NewPic.bmp
23/05/18	Test Decompress. Basically, the opposite to compression, we Compressed the file we made in the previous test and compared it to the original file.	It successfully decompressed and ended up being the same size as the original PIC.bmp image	PIC.bmp NewPic.bmp DeComp.bmp
28/05/18	Using the colour_printf function in colour.h, this test was used to see what colours would be the most visible and aesthetic for the purpose of our solution.	First, blue was used, however it was a bit difficult to see against the black background, so we used cyan instead. Although yellow was visible, it did not look the best, so we decided to pick green instead, which had a similar effect but was more aesthetic. Red was used for errors to suggest urgency.	N/A

Problems Encountered During Development

In General, we had a few major problems that arose through testing, which were fixed accordingly:

- For encryption, we were originally going to use a division-modulus algorithm, where the random key would be the divisor and the remainder would form the master key. Unfortunately, dividing the entire file as an integer is not exactly reasonable. After testing to confirm this consensus, we decided to instead go with a simple XOR process.
- We initially had some trouble with creating a system that could work both with arguments and an internal menu. This was mainly due the structure of the code. After testing alternative repetition and selection structures in main.c, we rewrote these in order to provide two methods of operation.
- We experienced some issues at first regarding the structure of all the huffman compression code. This was in part due to the lack of classes in C. eventually, we did find a workaround, but at the expense of code conciseness.

Conclusion

Overall, this solution has, through much hard work and testing, become a secure and reliable system for storing the Bepis Gola secret formula. It satisfies all requirements stated previously and has been completed within the time frame given. The functionality of the code is beyond what is required, as is the level of security and compression it provides. In conclusion, we the development team are very happy with the outcome of this project and hope it provides the security that Bepis Gola needs in order to operate, long into the future.

Appendices

Appendix A: Hand- Drawn Diagrams

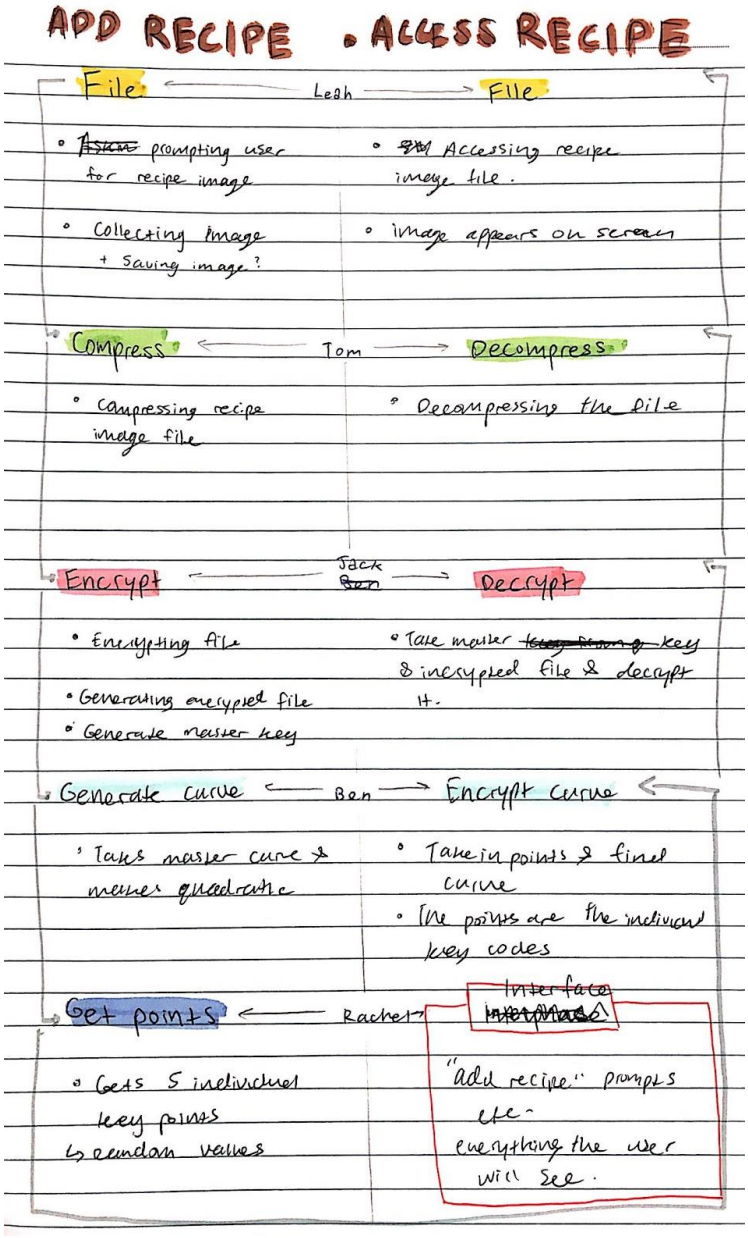

Fig. A.1: Draft flow chart and task allocation (Original Graphic by Leah Schirmer)

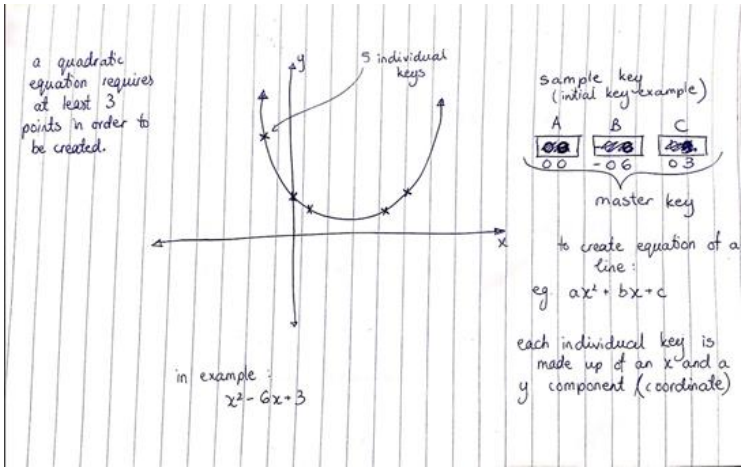

Fig. A.2: Graphical representation of how keys are used to unlock recipes using a parabola (Original Graphic by Rachel Birrel)

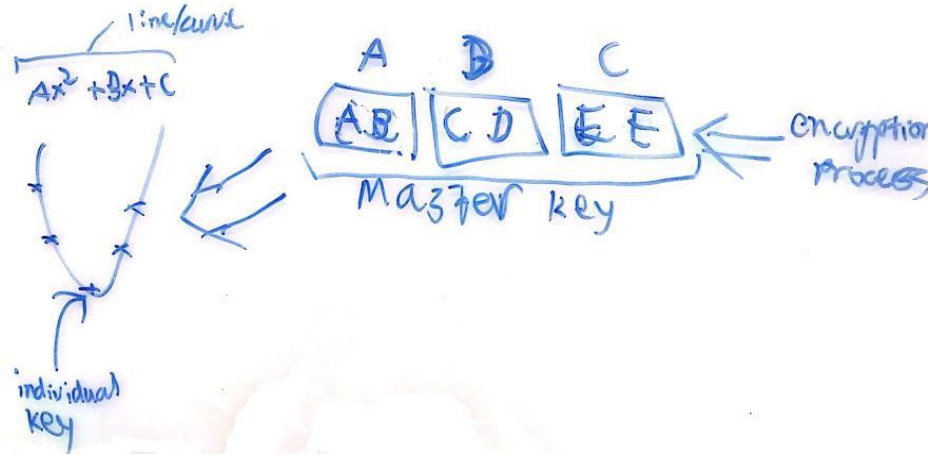

Fig. A.3: Original (basic) diagram of the key generation system (Original Graphic by Ben Kelly)

Appendix B: Design tools from section 4.4

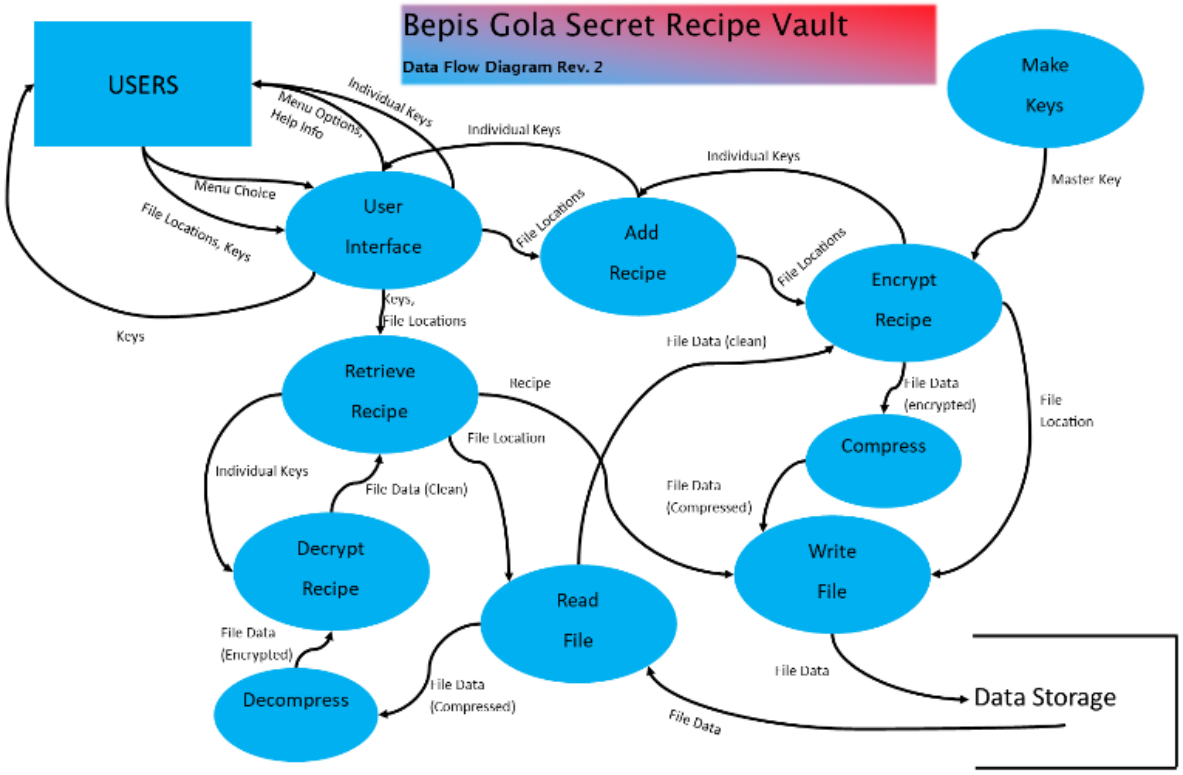

Fig. B.1: Data Flow Diagram (Original by Jack Christie)

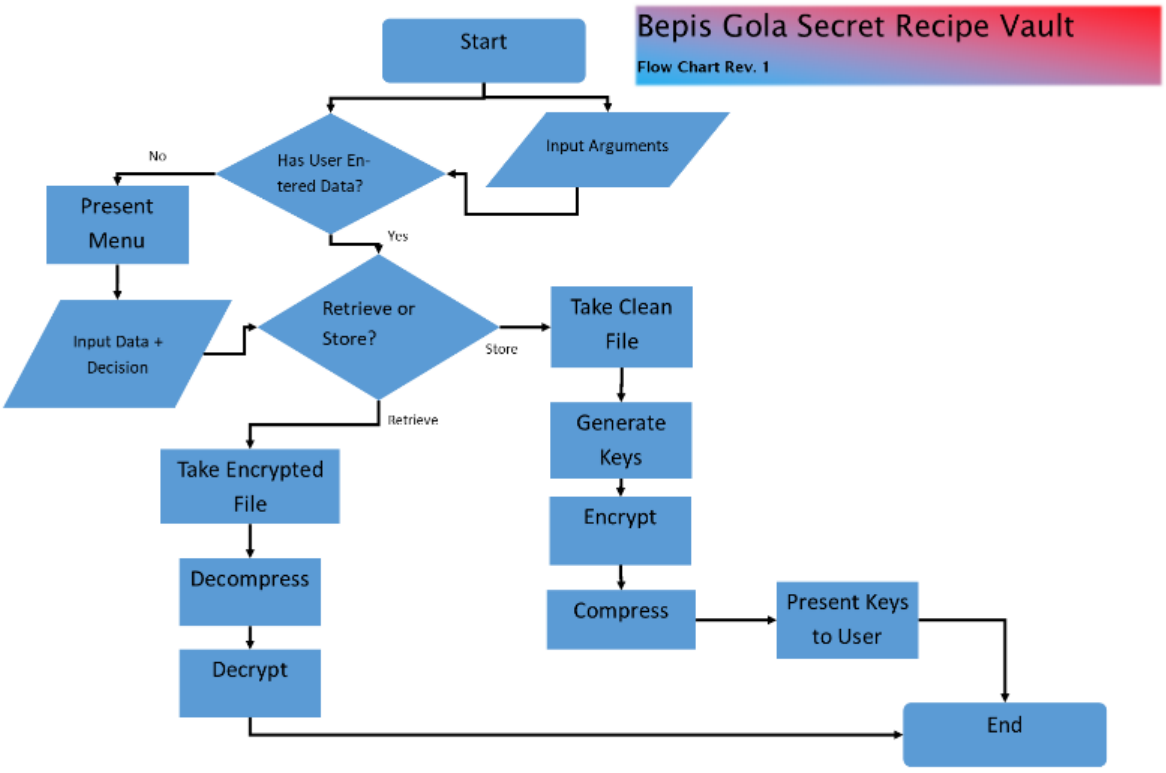

Figure B.2: Flowchart (Original Graphic by Jack Christie)

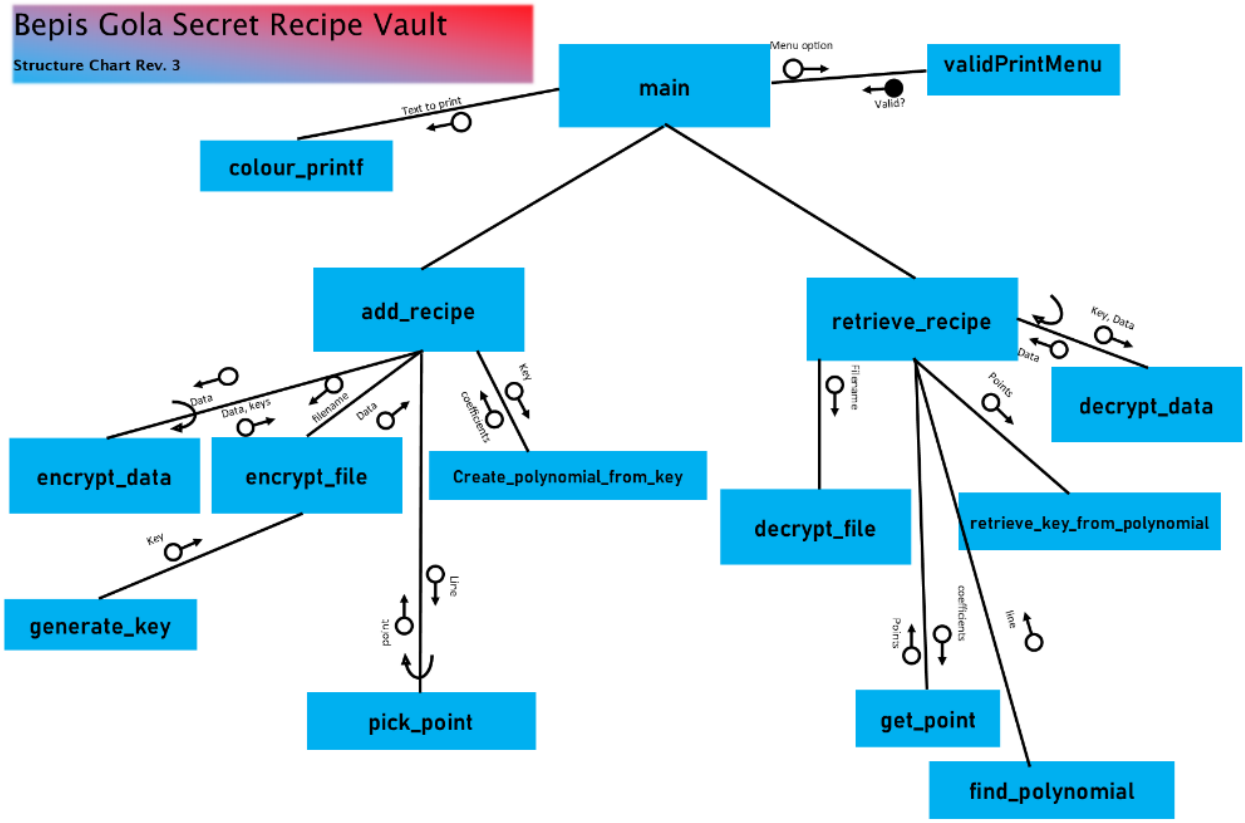

Figure B.3: Structure Chart (Original Graphic by Jack Christie)