Design Document on
A Password management Program

Objective: To create a **Password Manager**, which itself is secure too and can

(i) Take input of the password and a brief text description about it.
(ii) Encrypt the password according to an algorithm.
(iii) Save the encrypted passwords into a log file.
(iv) Read the log file according to the desired input when decrypting.
(v) Decrypt and prints the actual password when needed.

Report by: Group 6, Wednesday batch. (WED - 6)

Group members:

Akshat Malviya	(b17003)
Akshita Jain	(b17004)
Amit Chauhan	(b17006)
Vinay Kumar	(b17068)
Manpreet Singh	(b17091)

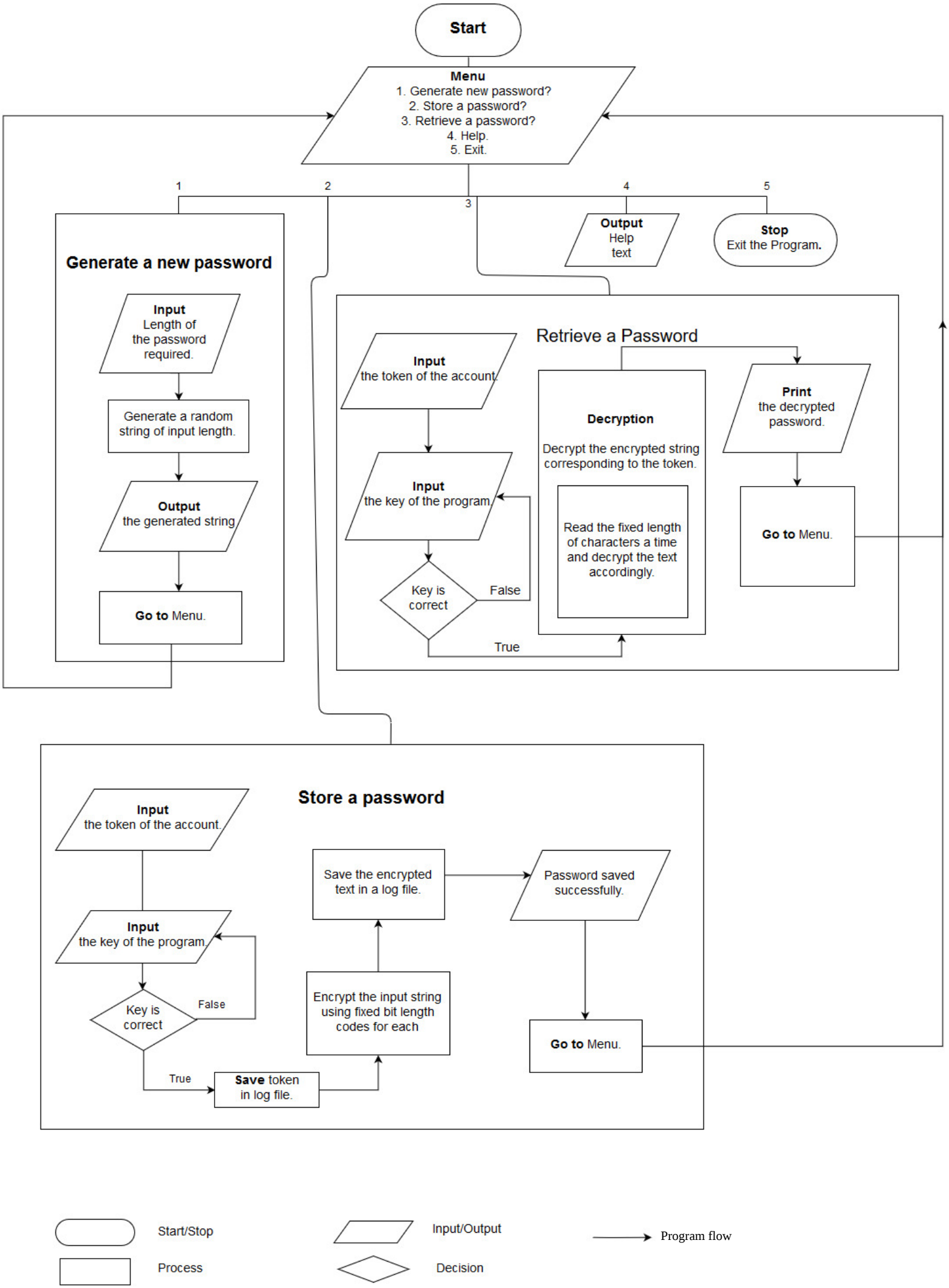
Why a Password Manager?

Suppose you're a person who has multiple password protected accounts at different places. Its obvious that remembering all the passwords will be a difficult task. Since, the accounts can contain many things which should be kept confidential, you'll prefer not to write the passwords anywhere, or say, tell anyone else and trust them to remember the passwords.

The attribute to this program by us is the feature of random password generation on the basis of the length required by the user. Basically, this feature works by generating random alphanumeric strings.

This is where a password manager becomes handy. A program which can "remember" the passwords. Password manager program stores the passwords after encrypting them into some form which isn't easy to decode (decrypt) and when needed, decrypts and tells the desired password.

Flowchart to understand the functions of the program



Specification of the Functions Used

(i) main() [Manpreet Singh]

- Prints the main menu, with the options to -
  - > Generate new Password.
  - > Store a password.
  - > Retrieve a stored password.
  - > Help text.
  - > Exit the Program.
- Calls functions/ Executes task according to the selected option from main menu.
- Contains Declarations of the input password array, its length, etc.

(ii) ranPass() [Amit Chauhan]

- Takes input of the length of the password required by the user.
- Generates random alphanumeric string according to the input length, using **random()** function which generates random number,
- Goes back to main().

(iii) encrypt() [Akshita Jain]

- Takes input of the token of account and password from the user.
- Asks for the Program key and asserts it using keyassert function.
- If key is *correct*, ***Fixed Bit Encoding** algorithm is used to ecrypt the characters and then stores the resulting string into a log file.
- If the key is *incorrect*, Asks for the key again, until the user enters correct Program Key.
- When done, returns to the main() function.

(iv) decrypt() [Akshat Malviya]

- Takes input of the token of account.
- Asks for the Program key and asserts it using keyassert() function.
- If key is *correct*, the logfile is read to find the encrypted string corresponding to the account token.
- Codes are read one-by-one (a fixed length of codes is read each time.) and converted to the corresponding character in the **prefixed charset.**

- When done, prints the original password corresponding to the token input by the used.

(v) help() [Akshat Malviya]

- Prints the help text.
  > Help Text: Has instructions for all the functions in the program.
  - Utility and how-to with example.

(vi) keyassert() [Vinay Kumar]

- When called by other functions, asserts that the Program key matches the actual program key, which is provided initially with the program. And is required to be remembered by the user.
- This is a boolean function, hence returns 0/1 corresponding to the equality of the Program key.

* Fixed Bit Length Encoding

- This encoding algorithm works by use of pre-fixed codes of fixed length, assigned to each character of the prefixed charset.

Example:

- Suppose we want to encode the string “ABCD”,
  Prefixed: charset = {‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’};
  Prefixed: charcodes:

A	=>	000
B	=>	010
C	=>	001
D	=>	011
E	=>	110
F	=>	101
G	=>	111
H	=>	100

- We use binary codes and assign them to characters in charset (Not in serial order, to make it a bit complex).
- Now, the string “ABCD” will be encoded (encrypted) as:

000010001011

- This is how a string is encoded. For decoding, the codes are read 3 digits at a time and corresponding characters are stored in the temporary output array, which, when completed, is printed to the user.

Encryption vs Encoding

Encoding is for maintaining *data usability* and can be reversed by employing the same algorithm that **encoded** the content, i.e. no key is used.
Encryption is for maintaining *data confidentiality* and requires the use of a key in order to return to the actual form.

We use an *encoding* algorithm *with a key*, as an *encryption* algorithm.

Conclusion:

- User can use the program to generate random strings of the desired length which can be used as password for various accounts.
- User doesn't need to remember passwords for multiple accounts instead, a single program key can be used to retrieve the desired password whenever needed.
- Person not knowing about the prefixed codes cannot say anything about the password. Hence the program is secure.