Y3 Introduction to C++
Cryptography Project

Ashley Robertson

Date: Jan 2015

School of Physics and Astronomy
University of Birmingham
Birmingham, B15 2TT

Contents

1 Introduction

The aim of this project is to construct a computer program using C++ coding language which encrypt and decrypt messages. One such a method of cryptography is "Symmetric Key Cryptography" which is going to be used.

2 Cryptography

Cryptography is a technique of private communication where the message that is send can only be read by the recipient. This technique which has roots in the history of humankind has had many forms and significant influence on historic events such as The Second World War. One might see cryptography as The Art of Secret Communication.

2.1 Symmetric Key Cryptography

One of the techniques of modern cryptography is Symmetric Key Cryptography (SKC). In this technique the sender and the receiver of the secret message will use identical password or key to encrypt and decrypt their messages.

In SKC the message and the encryption method can be send and received using communication options which can be eavesdropped by a third party. However, since the third party does not have access to the key, the secret message is not revealed (as long as the password can be kept secure).

The difficulty of this technique is to keep the key secure and secret from the third party. Therefore the method of communicating the secret keys must be safeguarded. On the other hand the security of the key is also dependent on how difficult it is to find the secret key by analysing the encrypted messages and the method of cryptography. This method of finding the secret key is called cryptanalysis.

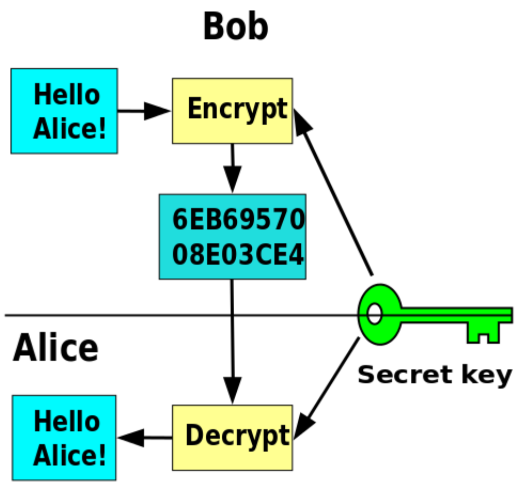


Figure 1: "Symmetric key encryption" by Phayzfaustyn - Own work. Licensed under CC0 via Wikimedia Commons

3 CryptoAsh

CryptoAsh is the name of the program that is developed in this project. This program aims to encrypt and decrypt the plain-text-files using a Symmetric Key Cryptography algorism that is developed in the project. This means that this program encrypt plain-text-files such as *.txt using a password which will also be used for the decryption.

CryptoAsh was developed using C++ language, Quincy 2005 compiler with FLTK verion 1.0 library, on a Microsoft Windows operating system. The compiled program is an ".exe" file which can be opened in a Microsoft Windows operating system.

The program starts with a welcome window which asks the user to select encryption or decryption functions or to close the program.

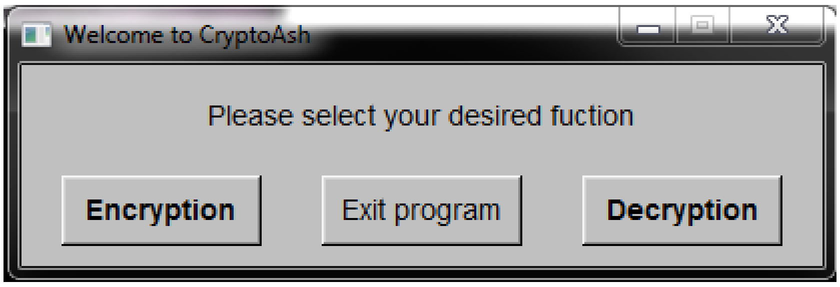


Figure 2: Welcome window in CryptoAsh

3.1 Encryption

when the user selects the encryption option, the welcome window will close, and the encryption window will be opened.

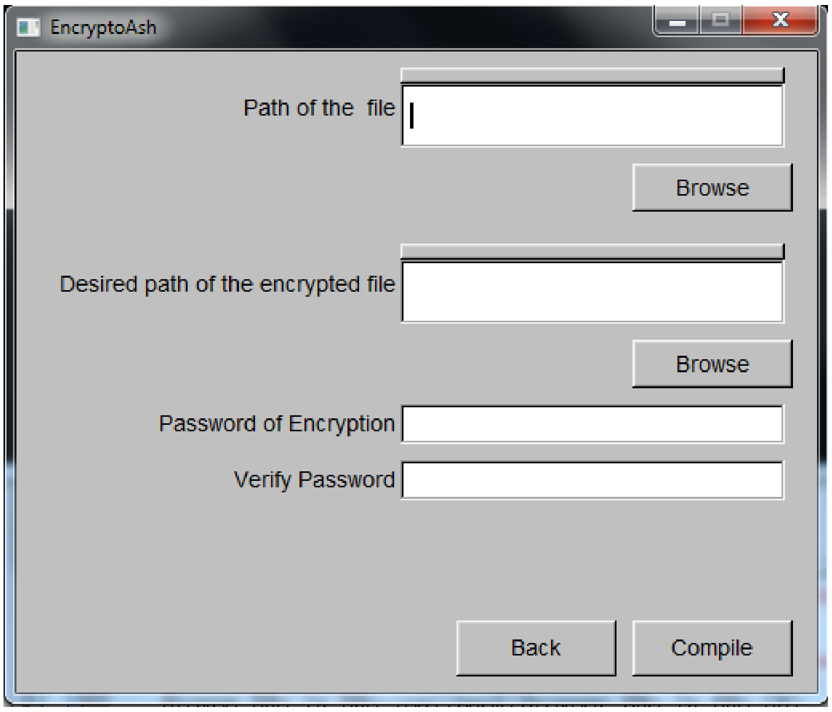
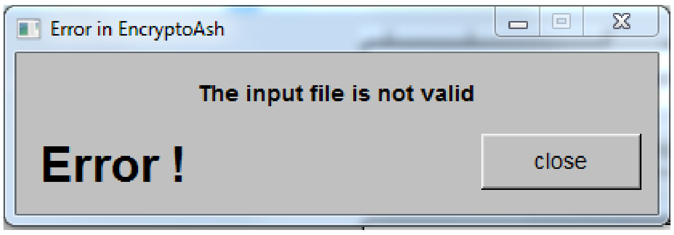


Figure 3: Encryption window in CryptoAsh

In this window, user should input the address of the file that is desired to be encrypted in the "Path of the file" section, and the address of the destination file, where the encrypted message will be stored in the "Desired path of the encrypted file". **It is important to note that if the "Desired path of the encrypted File" does not exist, the program will build that file, and if it does exist the program will <u>delete the content of that file and rewrite</u> the file with the encrypted message**.

If the user input files other than .txt, the program will still attempt on using the selected files. However, this program is not capable of encrypting files other than plain text files.


Error in EncryptoAsh

The input file is not valid

Error !

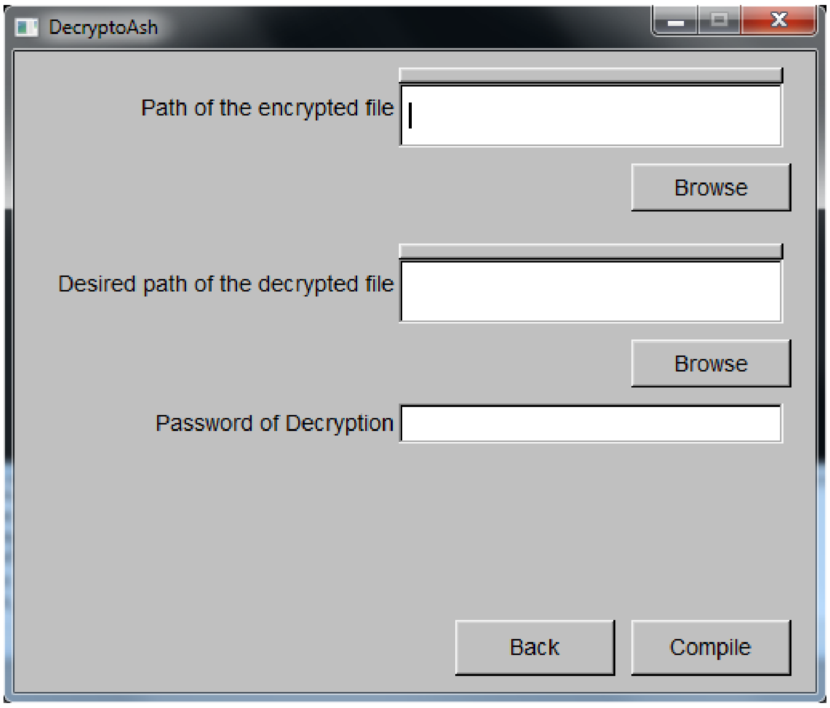
Figure 4: example of an error message in CryptoAsh

The program will also asks for a password, which is the key in the encryption process. To make sure that the user has input the right password, the program ask the user to re-enter the password in ”Verify Password”, and will only start the compiling if the entered password matches with the verified password. If these two do not match, the program will show an error message, explaining the problem.

When the files paths are entered and the password of encryption and its verification was input, the user may click the ”compile” button. the compile button will check whether the entered files can be opened, and the entered password matches with the verified password. If no problem was discovered, the compile button will then let the encryption compiling process to be run. On the other hand If the paths that are entered are not recognised, or they could not be opened or read, the program will prompt the user with an error message which inform the user of the problem, and does not compile, until the entered files are recognised.

3.2 Decryption

After selecting the ”Decryption” option in the welcome window, the welcome window will disappear, and the decryption window will appear.


DecryptoAsh

Path of the encrypted file

Desired path of the decrypted file

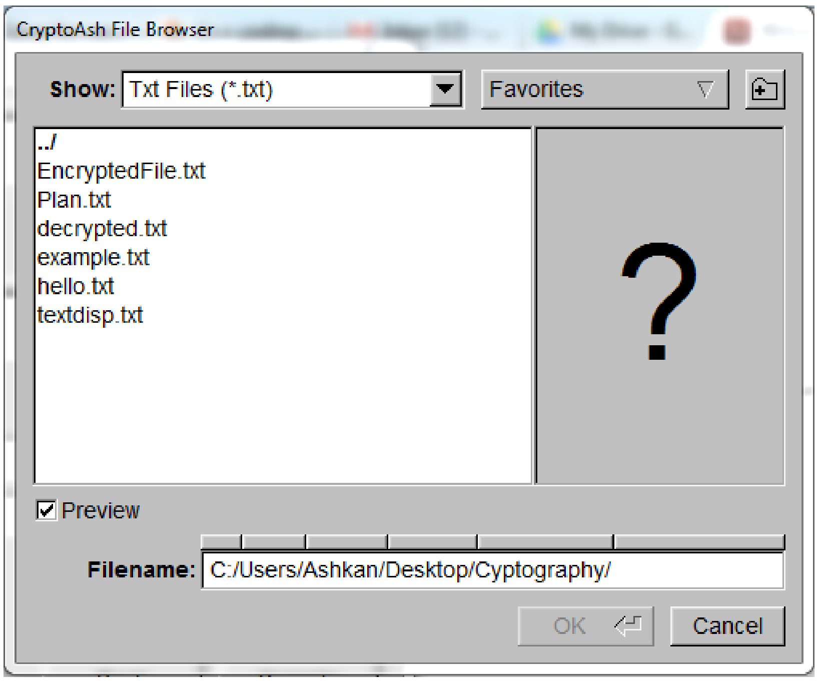
Password of Decryption

Figure 5: decryption window in CryptoAsh

In the similar fashion as the encrypted window, the decryption window will ask for the path of the encrypted file, path of the destination file, and the password of decryption which is the password used on the encrypted file.

After entering the file paths, and the password, the user may "compile" to decrypt the encrypted file into the selected destination. When the "compile" button is clicked, the program will check if the selected files can be opened. If they could be opened, the process continues, and if it could not, the procedure stops and the user is informed of the problem in the form of an error message.It should be noted that, if the selected destination does not exist the compiler will build it, and if it does, it will re-write it.

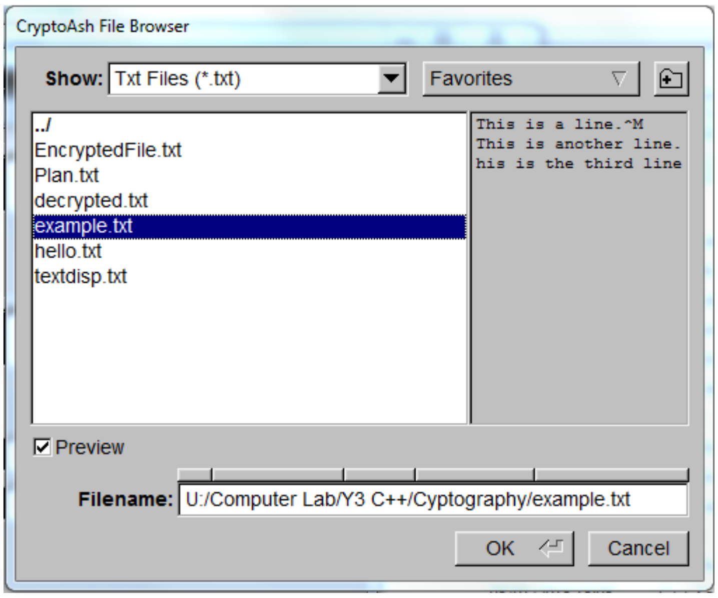
3.3 Browse Button



Filename: C:/Users/Ashkan/Desktop/Cyptography/

Figure 6: Browse window in CryptoAsh

For each path input widgets in this program, there is a browse button which helps the user find the path of their wanted file. When this window is opened, it shows the .txt files that are available in the directory which the program is run from. For example, if the program is saved in the desktop, the browser will initially show .txt files in the desktop. This filtering is designed to prompt the user to use .txt files which the program is designed to work with.
The user can search in different directories in the computer by chaning the "Filename" section at the bottom or by double cliking on "../" in the list of files, which opens the directory which was containing the previous one. Using the "Show:" drop-down menu, the user may change the filtering of directory "All File" or select a file type of their demand. However it is important to note that this program will not properly encrypt files other than plain text files.
After that the file is selected, the browse window will show a small preview of the file if the preview box is checked. When the file is selected and "OK" button was clicked, the file address will be copied into the the tab that the browse was called for.



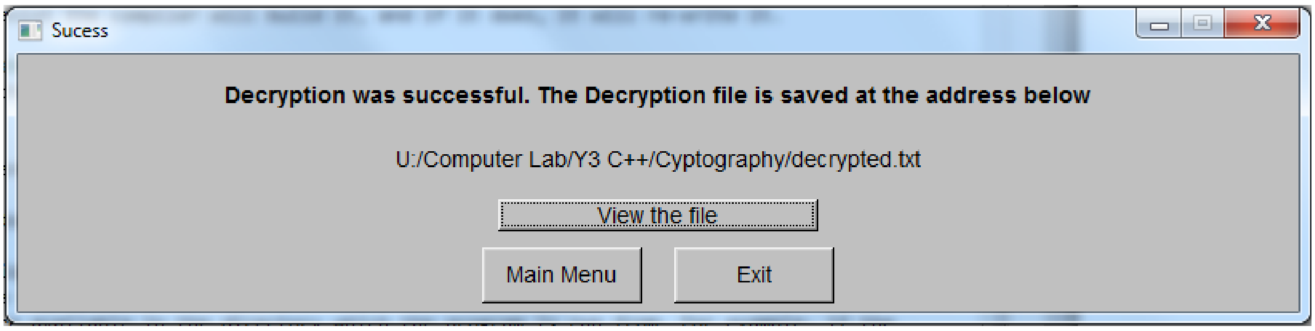
```
This is a line.^M
This is another line.
his is the third line
```

Filename: U:/Computer Lab/Y3 C++/Cyptography/example.txt

Figure 7: selecting file in the browse window

3.4 Compiling

When the compile button on either of the encryption and decryption windows is clicked, the compiler does some checks which are explained in encryption and the decryption sections. If the checks do not find a problem with the input data, cryptography will start. During the compiling, the data entering window will be closed, and a "Please Wait" message will be shown, until the compiling ends. However, Normally, the compile ends before the user can see the "please Wait" window. When the compile is ended, a success window will appear.

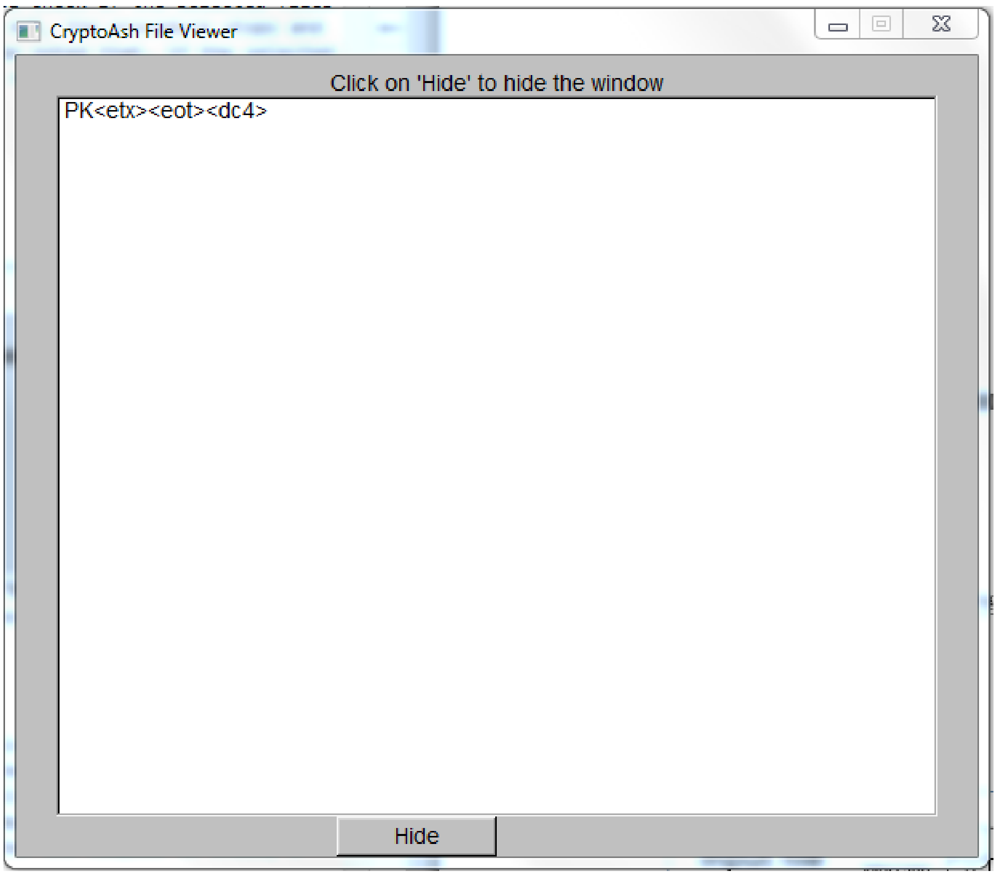


Decryption was successful. The Decryption file is saved at the address below

U:/Computer Lab/Y3 C++/Cyptography/decrypted.txt

Figure 8: success window in CryptoAsh

The success window will contain a success message, and the path to the destination file. It also gives you the option of opening the destination file in the program. The user may also exit the program by clicking on the "Exit" button, or return to welcome window by clicking on the "Main Menu" button



```
PK<etx><eot><dc4>
```

Figure 9: view file in the success window

3.5 Cryptography process

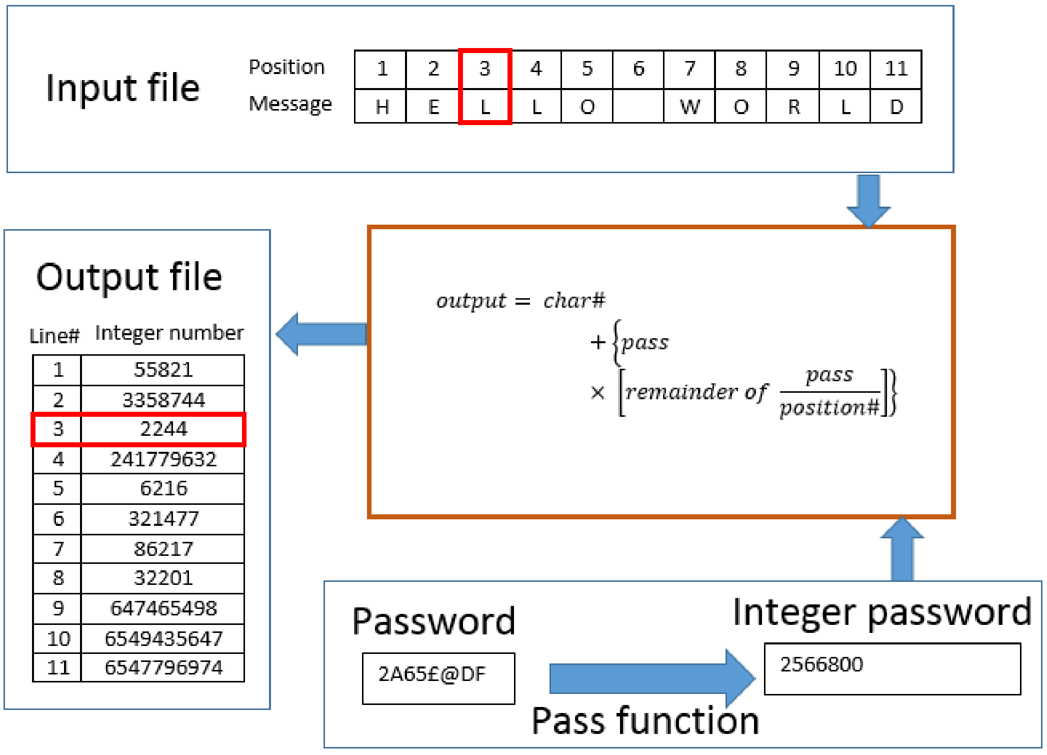


Figure 10: Encryption process in CryptoAsh

In order to encrypt a text, the program will take each character of the text message, and by using the encryption algorithm, converts it into a number and store it in the output file. for each character in the input file, the program will create a line which store the corresponding integer to that character.

The role of the encryption algorithms is to create a corresponding integer for each character. In this program, the encryption algorithm will take the ASCII equivalent of each character (called "*char*") and add position-pass integer to it where position-pass integer is a function of position of the character and the integer equivalent of the password (called "*Pass*"). Position-pass integer is the password multiplied by the remainder of the division of pass and position number of the character. (see equation 1)

$$\text{output integer} = \text{char} + \text{pass} \times [remainderof \frac{pass}{position}] \tag{1}$$

For equation 1 to work, the entered password must be converted to a unique integer number. For this purpose, a pass-function is used. The pass-function, converts a password into a unique integer by writing the ASCII equivalent of each character in the password alongside each other. therefore the longer the password is the larger the unique integer is. Intentionally, this program does not store the password in any means. This is to make sure that the encrypted message does not contain any clue which lead the message hacker to the correct password.

The decryption procedure will reverse the encryption process. To do this the decryption procedure use the password entered by the user, the line number, which is equivalent to the character number in the encryption process, and applies the reversed calculation to find the ASCII number of each encrypted characters. the ASCII numbers will be converted back to their corresponding characters, and then stored in the destination file.

3.6 Points of Improvement

(a) User Interface

The user-interface of the program could be improved by inserting some pictures, changing background colour, adding a favicon and etc. However, this point is more relevant in the stage of publishing the program.

(b) Encryption Method

Although Symmetric Key Cryptography is a modern type of cryptography and is considered to have a "Good" security, it is not the most secure of cryptography. Public and Private key cryptography is another type of modern cryptography which is commonly used for professional purposes. One might consider Public and Private Key Cryptography as a safer and more secure, but is a lot more sophisticated than the current algorithm used in this program, and also it requires more time to develop.

(c) Symmetric Key algorithm

The Symmetric Key algorithm used, was developed solely for this program and is not tested under different cryptanalysis techniques. Therefore one might say that the algorithm used is not safe from hackers. It might also be suggested that one can use ready developed cryptography library which empower the program with a professionally developed algorithms. However, for educational purposes of this program, the algorithm was developed in the project.

(d) limitation to plain-text-files

As already mentioned in the report, this program is limited to plain-text-files due to the design of the program. A developed version of this program can be achieved by making sure that the program is capable of encrypting and decrypting any type of file.

4 Appendix : Source Code

```
//Welcome to the CryptoAsh main source code
//written by Ashley Robertson
//Date: 18/01/2015
//Project Name:C++ Cryptography

#include <FL/Fl.H>
#include <FL/Fl_Window.h>
#include <FL/Fl_Box.h>
#include <FL/Fl_Button.H>
#include <FL/Fl_File_Input.H>
#include <FL/Fl_Secret_Input.H>
#include <FL/Fl_File_Chooser.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Text_Buffer.H>
#include <iostream>
#include <string>
#include <istream>
#include <ostream>
#include <fstream>
#include <cmath>

// FLTK widgets defined
Fl_Window *DEc;
Fl_Window *ENc;
Fl_Window *welcome;
Fl_Window *Compile_Win;
Fl_Window *waiting_Win;
Fl_Window *Txt_Win;
Fl_Box *box;
```

```cpp
Fl_Box *box_Error;
Fl_Box *box_Error1;
Fl_Box *box_Waiting;
Fl_Box *box_Compile;
Fl_Box *box_Compile_address;
Fl_Button *Encryption;
Fl_Button *Decryption;
Fl_File_Input *ENcFile;
Fl_File_Input *DEcFile;
Fl_File_Input *Actual_File;
Fl_File_Input *ENc_File;
Fl_Button *Decryption_Compile;
Fl_Button *Encryption_Compile;
Fl_Button *Decryption_back;
Fl_Button *Encryption_back;
Fl_Button *Exit;
Fl_Button *close_error_but;
Fl_Button *Browse_ENc_In_Dec;
Fl_Button *Browse_ACt_In_ENc;
Fl_Button *Browse_ENc_In_ENc;
Fl_Button *Browse_DEc_In_Dec;
Fl_Button *Compile_Success_Exit;
Fl_Button *Compile_Success_Welcome;
Fl_Button *Compile_Success_view;
Fl_Button *Txt_File_Hide;
Fl_Window *Error_Win;
Fl_Secret_Input *DEc_Pass;
Fl_Secret_Input *ENc_Pass;
Fl_Secret_Input *ENc_Pass_Check;
Fl_Text_Display   *Txt_File;
Fl_Text_Buffer *Txt_File_Buff;
const char *Result_Address;            //This is the address of the destination file of the compiler. Globally defined to connect information between functions

using namespace std;

//functions propotype
void Error_show(const char* txt);                             //show an error window showing txt input to the function
const char* file_chooser();                                   //opens a browser to selct file & returns address of the selected file
int IntPass( const char* password, int size);                    //converts a "const char" password into a long integer pasword
int Encryption_function (char TheCharcter, int position, int IntPass);  //decrypts a single char using its position and the integer pasword
char Decryption_Function (int TheNumber, int position, int IntPass); //encrypts a single char using its position and the integer pasword
void Compile_Message(const char* file_address, const char* message );   //shows a compiled message using the file address and the desired message
void View_File(const char* file_address);                     //view the plain txt file with the address provided to it
//calback functions
void Compile_decryption(Fl_Widget* w)              //Decryption compiling function
{
   ifstream Encrypted;                    //defining the files in the function
   ofstream Decrypted;
   //opening the Encryption and Decryption files & checking that they are opened properly
   Encrypted.open ( ENcFile->value(), std::ifstream::in);
   if (Encrypted.is_open())
   {
      Decrypted.open ( DEcFile->value(), std::ofstream::out);
      if (Decrypted.is_open())
      {
         //The decryption proccess starts from here
         DEc->hide();                     //Closing the compile page
         waiting_Win->show();             //prompting "Please Wait"
         int i = 0, c;
         char e;
         while (Encrypted >> c)               // loop getting single characters
         {
            i++;
            e = Decryption_Function (c, i, IntPass(DEc_Pass->value(), DEc_Pass->size())); //↩
                Each character is decrypted seperatly
            Decrypted.put(e) ;
         }
         Encrypted.close();
         Decrypted.close();                        // close files
         waiting_Win->hide();
         Compile_Message(DEcFile->value(), "Decryption was successful. The Decryption file ↩
             is saved at the address below" );
      }
      else Error_show("The input decrypted file is not valid");
   }
   else  Error_show("The input encrypted file is not valid");
}
void Compile_encryption(Fl_Widget* w)
{
   ifstream Actual;                    //defining the files in the function
   ofstream Encrypted;
   Actual.open ( Actual_File->value(), std::ifstream::in);
   if (Actual.is_open())
   {
      Encrypted.open ( ENc_File->value(), std::ofstream::out);
      if (Encrypted.is_open())
      {
         if (IntPass(ENc_Pass->value(), ENc_Pass->size()) == IntPass(ENc_Pass_Check->value()↩
             , ENc_Pass_Check->size()))
         {
            ENc->hide();
            waiting_Win->show();
            int i = 0, e;
            char c;
            while (Actual.get(c))           // loop getting single characters
            {
               i++;
               e = Encryption_function (c, i, IntPass(ENc_Pass->value(), ENc_Pass->size()));↩
                    //Each char is encrypted seperatly
               Encrypted << e << endl ;
            }
            Encrypted.close();
            Actual.close();                 // close file
            waiting_Win->hide();
            Compile_Message(ENc_File->value(), "Encryption was successful. The Encrypted ↩
                file is saved at the address below" );
         }
         else Error_show("Passwords do not match");
      }
      else Error_show("The input encryption file is not valid");
   }
   else  Error_show("The input file is not valid");
}
void Browser_ENc_In_Dec_CB(Fl_Widget*, void*)
{
   ENcFile->value(file_chooser());
}
void Browser_DEc_In_Dec_CB(Fl_Widget*, void*)
{
   DEcFile->value(file_chooser());
}
void Browser_ACt_In_ENc_CB(Fl_Widget*, void*)
{
   Actual_File->value(file_chooser());
}
void Browser_ENc_In_ENc_CB(Fl_Widget*, void*)
{
   ENc_File->value(file_chooser());
```

```
}
void show_decryption(Fl_Widget* w)
{
   DEc->show();
   welcome->hide();
}
void show_encryption(Fl_Widget* w)
{
   ENc->show();
   welcome->hide();
}
void Back_decryption(Fl_Widget* w)
{
   DEc->hide();
   welcome->show();
}
void Back_encryption(Fl_Widget* w)
{
   ENc->hide();
   welcome->show();
}
void close_error (Fl_Widget* w)
{
   Error_Win->hide();
}
void Exit_Calback(Fl_Widget* widget, void*)
{
   welcome->hide();
}
void Compiler_Success_Welcome(Fl_Widget*, void*)
{
   Compile_Win->hide();
   welcome->show();
}
void Compiler_Success_Exit(Fl_Widget*, void*)
{
   Compile_Win->hide();
}
void Compiler_Success_View(Fl_Widget*, void*)
{
   View_File(Result_Address);
}
void Txt_File_Hide_CB(Fl_Widget*, void*)
{
   Txt_Win->hide();
}
int main ()
{
   //Welcome window
   welcome = new Fl_Window(400, 100, "Welcome to CryptoAsh");
   welcome->begin();
   box = new Fl_Box(10, 10, 380, 30, "Please select your desired fuction");
   Encryption = new Fl_Button(20, 55, 100, 35, "Encryption");
   Encryption->labelfont(FL_HELVETICA_BOLD);
   Encryption->callback(show_encryption);
   Decryption = new Fl_Button(280, 55, 100, 35, "Decryption");
   Decryption->labelfont(FL_HELVETICA_BOLD);
   Decryption->callback(show_decryption);
   Exit = new Fl_Button(150, 55, 100, 35, "Exit program");
   Exit->callback(Exit_Calback);
   welcome->end();
   welcome->show();

   //Decryption window (DEc)
   DEc = new Fl_Window(500, 400, "DecryptoAsh");
   DEc->begin();
   ENcFile = new Fl_File_Input(240, 10, 240, 50, "Path of the encrypted file");
   Browse_ENc_In_Dec = new Fl_Button(385, 70, 100, 30, "Browse");
```

```
    Browse_ENc_In_Dec->callback(Browser_ENc_In_Dec_CB);
    DEcFile = new Fl_File_Input(240, 120, 240, 50, "Desired path of the decrypted file");
    Browse_DEc_In_Dec = new Fl_Button(385, 180, 100, 30, "Browse");
    Browse_DEc_In_Dec->callback(Browser_DEc_In_Dec_CB);
    DEc_Pass = new Fl_Secret_Input (240, 220, 240, 25, "Password of Decryption");
    Decryption_Compile = new Fl_Button(385, 355, 100, 35, "Compile");
    Decryption_Compile->callback(Compile_decryption);
    Decryption_back = new Fl_Button(275, 355, 100, 35, "Back");
    Decryption_back->callback(Back_decryption);
    DEc->end();

    //Encryption window (DEc)
    ENc = new Fl_Window(500, 400, "EncryptoAsh");
    ENc->begin();
    Actual_File = new Fl_File_Input(240, 10, 240, 50, "Path of the  file");
    Browse_ACt_In_ENc = new Fl_Button(385, 70, 100, 30, "Browse");
    Browse_ACt_In_ENc->callback(Browser_ACt_In_ENc_CB);
    ENc_File = new Fl_File_Input(240, 120, 240, 50, "Desired path of the encrypted file");
    Browse_ENc_In_ENc = new Fl_Button(385, 180, 100, 30, "Browse");
    Browse_ENc_In_ENc->callback(Browser_ENc_In_ENc_CB);
    ENc_Pass = new Fl_Secret_Input (240, 220, 240, 25, "Password of Encryption");
    ENc_Pass_Check = new Fl_Secret_Input (240, 255, 240, 25, "Verify Password");
    Encryption_Compile = new Fl_Button(385, 355, 100, 35, "Compile");
    Encryption_Compile->callback(Compile_encryption);
    Encryption_back = new Fl_Button(275, 355, 100, 35, "Back");
    Encryption_back->callback(Back_encryption);
    ENc->end();

    //Waiting window
    waiting_Win = new Fl_Window(250, 60, "Compiling in progress");
    waiting_Win->begin();
    box_Waiting = new Fl_Box(5, 5, 200, 55, "Please Waite");
    box_Waiting->labelfont(FL_HELVETICA_BOLD);
    box_Waiting->labelsize(32);
    waiting_Win->end();
    return Fl::run();
}
//Functions defined
void Error_show(const char* txt)
{
    //Error window
    Error_Win = new Fl_Window(400, 100, "Error in EncryptoAsh");
    Error_Win->begin();
    box_Error = new Fl_Box(10, 10, 380, 30, txt);
    box_Error->labelfont(FL_HELVETICA_BOLD);
    close_error_but = new Fl_Button(290, 50, 100, 35, "close");
    close_error_but->callback(close_error);
    box_Error1 = new Fl_Box(10, 50, 100, 40, "Error !");
    box_Error1->labelfont(FL_HELVETICA_BOLD);
    box_Error1->labelsize(30);
    Error_Win->end();
    Error_Win->show();
}
const char* file_chooser()
{
    // Create the file chooser, and show it
    Fl_File_Chooser chooser(".",                        // directory
                            "Txt Files (*.txt)",        // filter
                            Fl_File_Chooser::CREATE,    // chooser type
                            "CryptoAsh File Browser");  // title
    chooser.show();
    // Block until user picks something.
    //     (The other way to do this is to use a callback())
    //
    while (chooser.shown())
    {
        Fl::wait();
    }
```

```
   return chooser.value();
}
int IntPass( const char* password, int size)
{
   int IntPass = 0;
   int i;
   for ( i = 0; i < size ; i++)
   {
      IntPass = IntPass + (1000 ^ (i)) * (int)password[i];
   }
}
int Encryption_function (char TheCharcter, int position, int IntPass)
{
   //Encryption algorism
   int C = TheCharcter; //converting the character into ASCII number
   int i = 0;
   C = C + (IntPass * (IntPass % position));
   return C;
}
char Decryption_Function (int TheNumber, int position, int IntPass)
{
   TheNumber = TheNumber - (IntPass * (IntPass % position));
   char Replacement = TheNumber;
   return Replacement;
}
void Compile_Message(const char* file_address, const char* message )
{
   Result_Address = file_address;
   //Compile window
   Compile_Win = new Fl_Window(800, 160, "Sucess");
   Compile_Win->begin();
   box_Compile = new Fl_Box(10, 10, 780, 30, message);
   box_Compile->labelfont(FL_HELVETICA_BOLD);
   box_Compile_address = new Fl_Box(10, 50, 780, 30, file_address);
   Compile_Success_view = new Fl_Button(300, 90, 200, 20, "View the file");
   Compile_Success_Exit = new Fl_Button(410, 120, 100, 35, "Exit");
   Compile_Success_Welcome = new Fl_Button(290, 120, 100, 35, "Main Menu");
   Compile_Success_Welcome->callback(Compiler_Success_Welcome);
   Compile_Success_Exit->callback(Compiler_Success_Exit);
   Compile_Success_view->callback(Compiler_Success_View);
   Compile_Win->end();
   Compile_Win->show();
}
void View_File(const char* file_address)
{
   Txt_Win = new Fl_Window (600, 500, "CryptoAsh File Viewer");
   Txt_Win->begin();
   Txt_File = new Fl_Text_Display(25, 25, 550, 450, "Click on 'Hide' to hide the window");
   Txt_File_Buff = new Fl_Text_Buffer();
   Txt_File->buffer(Txt_File_Buff);
   Txt_File_Buff->loadfile(file_address);
   int pos = 0;
   while (Txt_File_Buff->findchar_forward(pos, '\r', &pos))
   {
      Txt_File_Buff->remove(pos, pos + 1);
   }
   Txt_File_Hide = new Fl_Button(200, 475, 100, 25, "Hide");
   Txt_File_Hide->callback(Txt_File_Hide_CB);
   Txt_Win->end();
   Txt_Win->show();
}
```

Source Code 1: main source code of CryptoAsh.exe.