A Project Report

on

File Encryptor in C++

Submitted in partial fulfillment of the requirement of PROJECT

PROJECT-I (BCE3009)

of

Bachelor in Computer Engineering

**Submitted to**

****

Purbanchal University

Biratnagar, Nepal

**Submitted By**

Aayush Kumar Mallik (731744)

Saraswoti Rokaya (731763)

Salim Shrestha (731759)

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

A Project Report

on

File Encryptor in C++

Submitted in partial fulfillment of the requirement of PROJECT

PROJECT- I (BCE3009)

of

Bachelor in Computer Engineering

**Submitted to**

Purbanchal University

Biratnagar, Nepal

**Submitted By**

< Saraswoti Rokaya > <731763>

< Salim Shrestha > <731759>

< Aayush Kumar Mallik > < 731744>

**Project Supervisor**

**Mr. Kiran Khanal**

**Senior Assistant Professor**

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

**Abstract**

In today's digital world, keeping data safe and private is very important. With more files being stored and shared online, there is a big risk of information getting stolen or misused. To help protect this data, our project, titled **"File Encryptor in C++"**, focuses on building a simple and effective tool that can lock and unlock files using basic encryption. The Caesar Cipher is one of the oldest and easiest methods of encryption. It works by shifting each letter in a file by a fixed number of positions in the alphabet. For example, if we shift by 3, 'A' becomes 'D', 'B' becomes 'E', and so on. Developed in C++, the project highlights essential programming concepts like file handling, error detection, and user interaction. It also includes a graphical user interface built using the WinBGIm graphics library, providing sound and visual feedback for a better user experience. Though this method is not very secure for advanced use, it is a great way to understand the basic idea of how encryption works. We chose to implement this in **C++** because it is a powerful and fast programming language that gives good control over file handling and system operations.

Our program allows users to select a file and apply the Caesar Cipher to either encrypt or decrypt its contents. The tool supports plain text files and processes every character, including handling spaces, punctuation, and numbers properly. It reads the file content, applies the encryption logic, and saves the result to a new file. The program also checks for errors like missing files, invalid keys, or empty input. The main goal of this project is to help students and beginners understand how basic encryption works and how to implement it in real programs. While Caesar Cipher is not used in real-life secure systems, this project provides a solid foundation for learning more advanced encryption methods in the future. Overall, this project combines programming and security concepts to offer a fun and educational experience.

**ACKNOWLEDGEMENT**

We extend our heartfelt gratitude to all those who supported us in completing this project titled **"File Encryptor in C++"**. First and foremost, we sincerely thank our supervisor, **Mr. Kiran Khanal**, for his valuable guidance and continuous encouragement. His knowledge and feedback were crucial throughout the development process.

We also thank the faculty members and staff of the **Department of Computer Engineering** for providing us with the necessary resources and a supportive environment. Our special thanks go to our classmates and friends for their suggestions and moral support. Lastly, we are grateful to our families for their patience and unwavering encouragement during this journey.

**DECLARATION**

We hereby declare that the project report entitled **“File Encryptor in C++”**, submitted in partial fulfillment of the requirements for the degree of **Bachelor of Engineering in Computer Engineering**, is the result of our original work carried out under the supervision of **Mr. Kiran Khanal**. This work has not been submitted previously for the award of any degree, diploma, or similar title at any other institution or university. In accordance with academic and ethical standards, proper acknowledgements have been given wherever the work of others has been referenced.

**Salim Shrestha**

**Saraswoti Rokaya**

**Aayush Kumar Mallik**

**Date: 2025**

**SUPERVISOR’S APPROVAL**

This is to certify that the major project entitled “**File Encryptor in C++”** undertaken and successfully demonstrated by **Salim Shrestha, Saraswoti Rokaya and Aayush Kumar Mallik**, has been completed under my guidance. This project is submitted as partial fulfillment of the requirements for the degree of **Bachelor of Engineering in Computer Engineering** under **Purbanchal University**. Throughout the duration of the project, the students have shown dedication, strong technical skills, and a clear understanding of the subject matter. Their performance during the development and presentation of the project reflects their readiness to take on professional responsibilities in the field. I hereby approve this project for certification by the concerned authority.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Mr. Kiran Khanal**

Senior Assistant Professor

**Date: 2025**

# CERTIFICATE FROM DEPARTMENT

This is to certify that, following the Supervisor’s Approval and Examiners’ Acceptance, the project entitled “**File Encryptor in C++”**, submitted by **Salim Shrestha, Saraswoti Rokaya and Aayush Kumar Mallik**, has been officially approved as a partial fulfillment of the requirements for the degree of Bachelor **of Engineering in Computer Engineering** under **Purbanchal University**. The department acknowledges the students’ efforts and successful completion of the project.

We commend their work and wish them continued success in all their future endeavors.

**[Official Signature plus Stamp]**

**Er. Subash Rajkarnikar**

Head of Department,

Department Of Computer Engineering,

Kantipur City College

**Date: 2025**

# Table Of Contents

[CERTIFICATE FROM DEPARTMENT 7](#_Toc203772135)

[List Of Figures 10](#_Toc203772136)

[1. Introduction 11](#_Toc203772137)

[**Overview 11**](#_Toc203772138)

[**Problem statement 12**](#_Toc203772139)

[**Features 12**](#_Toc203772140)

[Objective 12](#_Toc203772141)

[**Confidentiality: 12**](#_Toc203772142)

[**Data Integrity: 12**](#_Toc203772143)

[**Access Control: 13**](#_Toc203772144)

[**Secure Storage: 13**](#_Toc203772145)

[**Significance 13**](#_Toc203772146)

[2. Scope and Limitation 13](#_Toc203772147)

[**Scope 13**](#_Toc203772148)

[**Limitation 14**](#_Toc203772149)

[3. Methodology 14](#_Toc203772150)

[**Software Development Life cycle 14**](#_Toc203772151)

[4. Technologies and Tools 16](#_Toc203772152)

[**Technologies and Tools used for the File Encryptor Project 16**](#_Toc203772153)

[5. System Design 18](#_Toc203772154)

[**System Architecture 18**](#_Toc203772155)

[**Flowchart 19**](#_Toc203772156)

[6. Object Oriented 20](#_Toc203772157)

[**Class Diagram 20**](#_Toc203772158)

[**Use Case Diagram 21**](#_Toc203772159)

[**Activity Diagram 22**](#_Toc203772160)

[**Sequence Diagram 23**](#_Toc203772161)

[7. System Development and Implementation 24](#_Toc203772162)

[**Programming Platform 24**](#_Toc203772163)

[**Operating Environment 24**](#_Toc203772164)

[8. Assignment of Roles and Responsibilities 25](#_Toc203772165)

[**Member Name 25**](#_Toc203772166)

[**Role & Responsibilities 25**](#_Toc203772167)

[**Saraswati Rokaya 25**](#_Toc203772168)

[**Salim Shrestha 25**](#_Toc203772169)

[**Aayush Kumar Mallik 25**](#_Toc203772170)

[9. Testing and Debugging 26](#_Toc203772171)

[10. Conclusion 27](#_Toc203772172)

[11. References 28](#_Toc203772173)

# Introduction

In the digital era, securing sensitive information has become more important than ever. The File Encryptor in C++ project is designed to provide a reliable way to protect confidential data by converting readable files into an unreadable format using encryption algorithms. This ensures that even if unauthorized users gain access to the files, they cannot understand or misuse the information without the correct decryption key or password.

The project is not only a demonstration of basic cryptography but also serves as a practical example of file handling, UI design using WinBGIm, and implementing basic security protocols.

## Overview

The **File Encryptor** project is a desktop-based application developed in C++ that allows users to securely encrypt and decrypt files, helping protect sensitive data from unauthorized access. In an era of increasing concern over data privacy and cybersecurity, this tool provides a practical and educational solution for safeguarding personal or confidential information using a basic encryption algorithm—the Caesar Cipher.

Users can select a file through a graphical interface, choose an operation (encrypt or decrypt), and the program will transform the file content into an unreadable format (cipher text), which can only be restored using the correct decryption method. The system includes key features such as file validation, visual feedback, error handling, and operation history logging to ensure a smooth and user-friendly experience.

Designed for simplicity and clarity, this project demonstrates the core concepts of file encryption and offers a strong foundation for students and beginners to understand and experiment with cryptographic principles.

## Problem statement

In the current digital landscape, data is frequently shared and stored across various platforms, often without adequate protection. Many users, especially individuals and small organizations, rely on basic file storage methods that do not include any form of encryption. As a result, sensitive files are vulnerable to unauthorized access, data breaches, and cyber-attacks. Existing commercial encryption tools may be costly, complex, or require internet connectivity and advanced technical knowledge, making them less accessible to general users or students.

Moreover, some open-source or built-in file encryption solutions offer limited customization, lack transparency in their encryption processes, or provide minimal control over key management. This creates a significant gap for users who need a lightweight, efficient, and easy-to-use encryption tool for protecting personal or confidential data. There is a clear need for a standalone, platform-independent file encryptor that prioritizes security, simplicity, and control addressing the shortcomings of current systems while being accessible for educational and practical use.

## Features

**File Encryption & Decryption**  
Encrypts and decrypts files using the Caesar Cipher with a fixed shift value.

**Graphical User Interface (GUI)**  
User-friendly interface built with graphics.h, allowing navigation through menus with mouse clicks.

**File Selection Dialog**  
Users can browse and select files using a standard Windows file dialog box.

**Loading Animation**  
Displays a loading animation during encryption/decryption to indicate progress.

**Sound Feedback**  
Plays success or error sounds upon completion of operations for better user interaction.

**History Logging**  
Logs every encryption or decryption action along with a timestamp in a history.txt file.

**Error Handling**  
Displays messages for invalid files, missing input, or operation failure.

**Visual Confirmation**  
Message boxes and on-screen prompts confirm each action and outcome.

**Supports Text and Binary Files**  
Can handle different file types, ensuring flexibility in usage.

**Lightweight and Standalone**  
Does not rely on external libraries (except graphics.h) and runs as a single executable.

# Objective

### Confidentiality:

Ensure that the contents of a file cannot be understood by unauthorized users.

Encryption transforms readable data (plaintext) into unreadable data (ciphertext).

### Data Integrity:

Prevent undetected modification of the file during storage or transmission.

While encryption alone doesn’t guarantee integrity,

### ****Access Control:****

Only authorized users with the correct decryption key can access the file’s original content.

### ****Secure Storage:****

Protect sensitive data stored on disk, such as user credentials, personal information, or business documents.

**Secure Transmission:**

Ensure the file remains protected during network transmission, preventing eavesdropping.

# Scope and Limitation

## Scope

* Provide a **simple yet functional** tool to encrypt and decrypt files.
* Help users **protect files** from unauthorized access.
* Implement the **Caesar Cipher algorithm** for:
  + Text files
  + Binary files
* Allow users to:
  + **Select a file**
  + Choose an operation: **Encryption** or **Decryption**
  + Generate a **new output file**
* Built using **C++** with the **WinBGIm graphics library**.
* GUI features include:
* **File selection dialog**
* **Loading animations**
* **Sound effects**
* **History logging**
* Target audience:
  + **Students and beginners** learning file handling and encryption
* Educational value:
  + Demonstrates real-world use of **cryptographic operations**
* Current limitations:
  + Only uses **Caesar Cipher**
  + Works only on **Windows**
* Future extension possibilities:
  + **Stronger encryption algorithms**
  + **Password protection**
  + **Cross-platform support**

## Limitation

* The program does not support network or cloud-based file encryption.
* It uses a fixed encryption algorithm and does not allow algorithm switching.
* Does not store or manage keys securely key/password must be remembered by the user.
* It may not work properly with very large files due to memory usage.
* Does not provide protection against advanced attacks like side-channel or brute-force unless combined with strong passwords.

# Methodology

The **File Encryptor** in C++ was developed using a modular and iterative approach. The project followed Agile principles, breaking the work into small sprints focusing on core features like encryption logic, file handling, and the graphical user interface. Early prototypes were built to test Caesar Cipher functionality before integrating advanced features such as file selection dialogs, sound effects, and history logging. Each module was tested individually, and peer feedback was used to refine the design. This method ensured a clear development process and a reliable, user-friendly final product.

## 2.1 Development Approach: Agile + Modular Prototyping

The development of the File Encryptor in C++ project followed a structured, modular approach incorporating Agile principles and iterative prototyping. The methodology was tailored to the nature of system utility software development, emphasizing quick testing, continuous feedback, and incremental functionality. The project was developed in sprints, each focusing on a specific functional component such as encryption logic, user interface, or file handling.

**Key Phases in Development:**

1. **Requirement Planning:**

* Core functionalities such as Caesar Cipher encryption/decryption, file I/O operations, and UI interaction were identified and scoped.
* Features were categorized as:
* **Core Features**: File selection, Caesar Cipher logic, encryption/decryption execution, file output, success/error handling.
* **Stretch Features**: Graphical user interface using WinBGIm, history logging, sound feedback, file validation.

1. **Rapid Prototyping:**

* Early prototypes focused on verifying Caesar Cipher logic using console-based input/output.
* Initial file reading/writing mechanisms were implemented and tested on small .txt files.
* Prototypes helped validate the shift-key logic and ensured that special characters remained unaffected.

1. **Incremental Feature Development:**

* Components were developed incrementally in logically isolated modules:
* **Encryption Module**: Handles Caesar Cipher transformations with support for both uppercase and lowercase alphabets.
* **File Handler:** Manages reading from and writing to user-specified files in binary mode.
* **UI Layer:** Developed using the WinBGIm graphics library to allow user navigation, file selection, and visual feedback.
* **Logger Module**: Appends each successful operation (with timestamp) to a history.txt file.
* C++ namespaces and classes were used to ensure clean code separation and maintainability.

1. **Testing and Feedback:**

* Each module was tested independently for:
* File access reliability.
* Correctness of encryption/decryption output.
* Response to edge cases (empty files, non-text characters).
* Peer testing was conducted to ensure the GUI was user-friendly and logically structured.
* Feedback led to enhancements in error messages and improvements in visual animations.

1. **Integration and Optimization:**

* All independent modules were combined into a unified application with a main menu and user interaction loop.
* Load animations, click-detection logic, and file dialogs were integrated seamlessly.
* Audio cues and loading indicators were added to improve user experience and accessibility.

1. **Final Polishing and Maintenance:**

* Final version was cleaned of bugs, optimized for basic performance, and packaged for demonstration.
* Codebase was documented with comments and structured headers for maintainability.
* Future improvements may include:
* Password-based encryption.
* Support for multiple algorithms.
* Cross-platform GUI integration (beyond graphics.h).

# Technologies and Tools Used

## Technologies and Tools used for the File Encryptor Project

|  |  |  |
| --- | --- | --- |
| **SN** | **TOOLS** | **PURPOSE** |
| 1 | C++ | Core programming language |
| 2 | VS Code, DEV C | Write, debug, and run code |
| 3 | Compiler (GCC, MSVC) | Turn code into executable |
| 4 | Standard Libraries | File I/O, encryption logic |
| 5 | Operating system | Any major OS (Windows/Linux/macOS) |

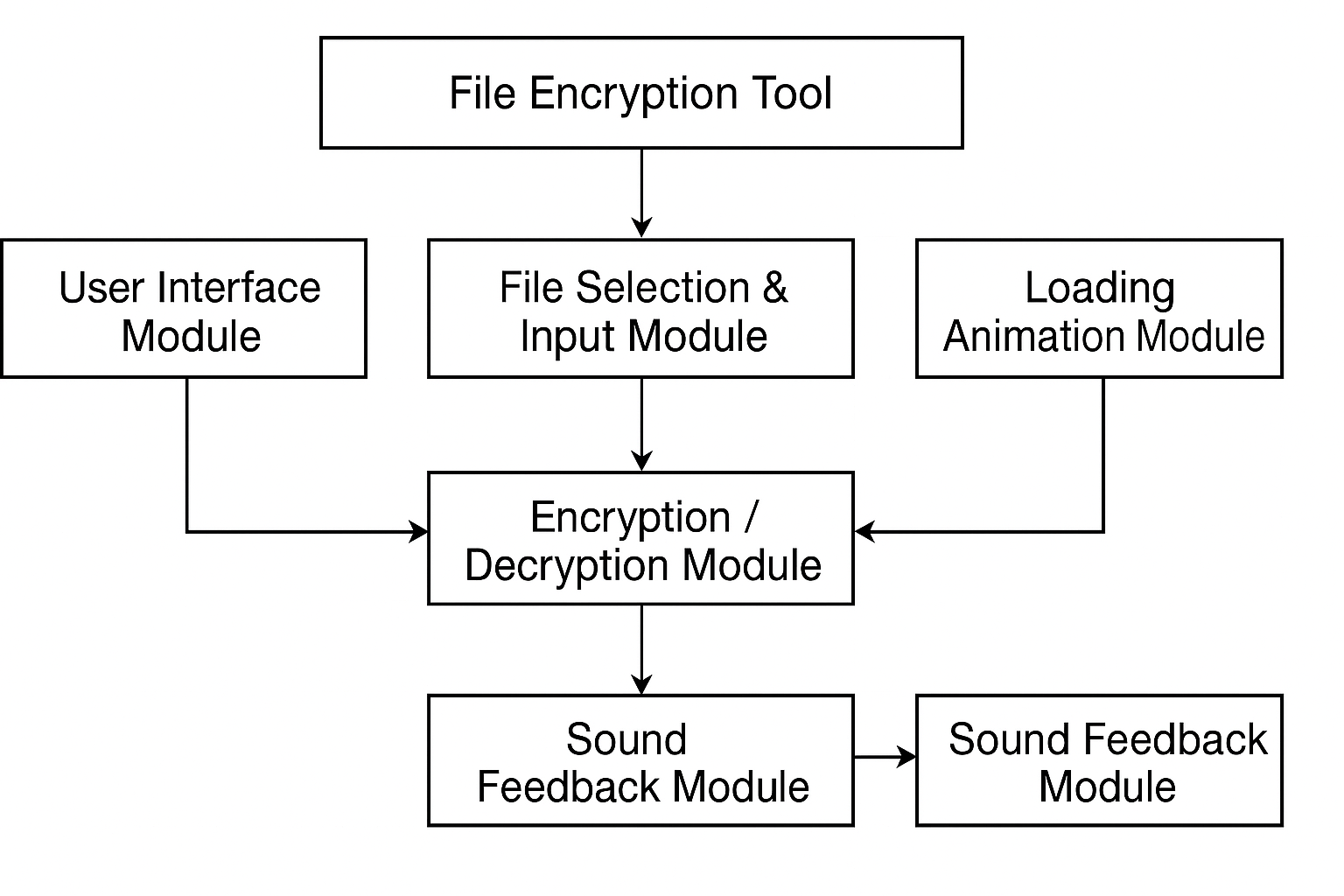
Fig 2: Table of file encryptor in c++

# System Design

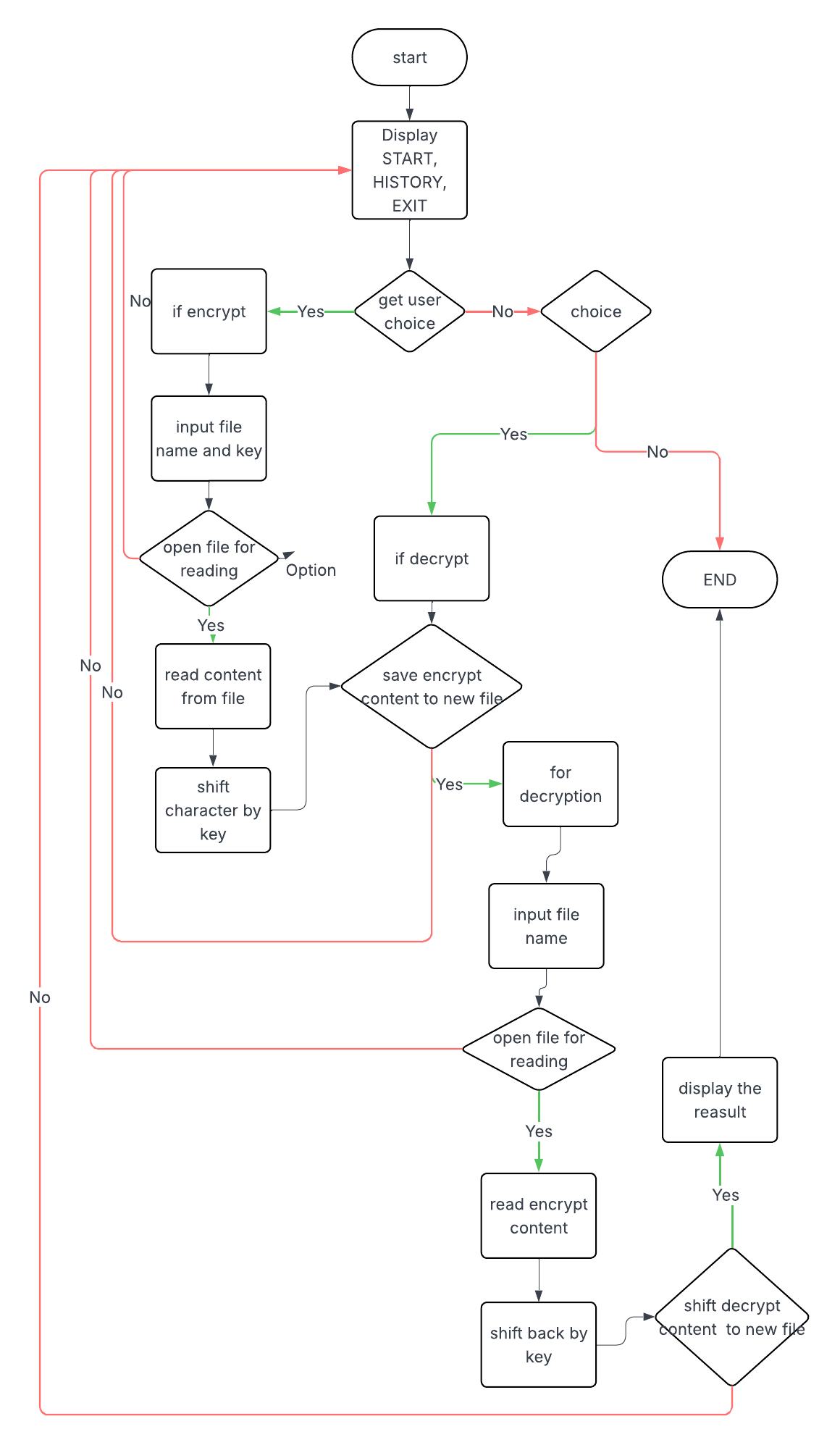
## System Architecture

The system follows a modular architecture:

* **UI Layer**: Built with WinBGIm, provides main menu, animations, file dialog.
* **Core Logic**: Handles Caesar Cipher encryption and decryption.
* **File Handling**: Reads input and writes output securely.
* **Logging Module**: Records actions in history.txt.
* **Feedback**: Provides user interaction via sound and visual effects.



## Flowchart



**Fig 3:** [Flowchart of file encryptor in c++](https://lucid.app/lucidchart/827a8802-6e0b-474f-bcde-535bc5019c33/edit?invitationId=inv_6a9132b9-894f-40ff-a038-d0c088e596b7)

# Yha sam ma here ------------

# Object Oriented

## Class Diagram

The **Caesar Cipher** is one of the oldest and simplest encryption algorithms. It works by shifting each letter in the plaintext by a fixed number of positions in the alphabet. For example, with a shift of 3, the letter 'A' becomes 'D', 'B' becomes 'E', and so on. If the shift goes beyond 'Z', it wraps around to the beginning of the alphabet, so 'Z' would become 'C'. To decrypt the message, the letters are shifted back in the opposite direction by the same number. Characters that are not letters, such as numbers or punctuation, are typically left unchanged. The Caesar Cipher is a form of substitution cipher and is very easy to break using brute-force or frequency analysis techniques.



Fig 4: class diagram of file encryptor in c++

## Use Case Diagram

In the use case diagram for the project **"File Encryptor in C++ "**, the main actor is the **User**, who interacts with the application through a simple command-line interface. The user can perform several key actions. First, the user can **Input a File**, which is the file that needs to be encrypted or decrypted. Next, the user chooses to either **Encrypt File** or **Decrypt File**, depending on the task. For encryption, the user provides a **Shift Key (a number)**, which is used by the Caesar Cipher algorithm to shift characters in the file content. After this, the system performs the **Apply Caesar Cipher** use case, where each letter or byte of the file is shifted forward or backward in the alphabet based on the key. The encrypted file is then **Saved to Output File**, which the user can store securely. If decryption is selected, the process reverses: the system reads the encrypted file, uses the shift key to reverse the encryption, and **Displays or Saves the Decrypted File.** Additional optional use cases include **View File Content, Check for Errors** (e.g., wrong key or missing file), and **Exit Program**. This simple structure makes it easy for anyone to understand and use the program securely and effectively.



Fig 5:- use case diagram of file encryptor in c++

## Activity Diagram

The **activity diagram of file encryption in C++** represents the logical flow of actions involved in encrypting a file. The process begins with the user starting the program and entering the filename of the file to be encrypted, along with an encryption key. The system first checks whether the specified file exists. If the file is not found, an error message is displayed and the process terminates. If the file does exist, the program proceeds to read the contents of the file. Once the content is successfully read, the encryption module uses the provided key to encrypt the data using a selected encryption algorithm (such as XOR, AES, or a custom method). After encryption, the user is prompted to input a name for the output file. The program then writes the encrypted content to this new file. Finally, a success message is displayed to inform the user that the encryption process has been completed successfully. This activity diagram effectively outlines the step-by-step control flow from receiving input to saving the encrypted output and ensures error handling for common issues like missing input files.

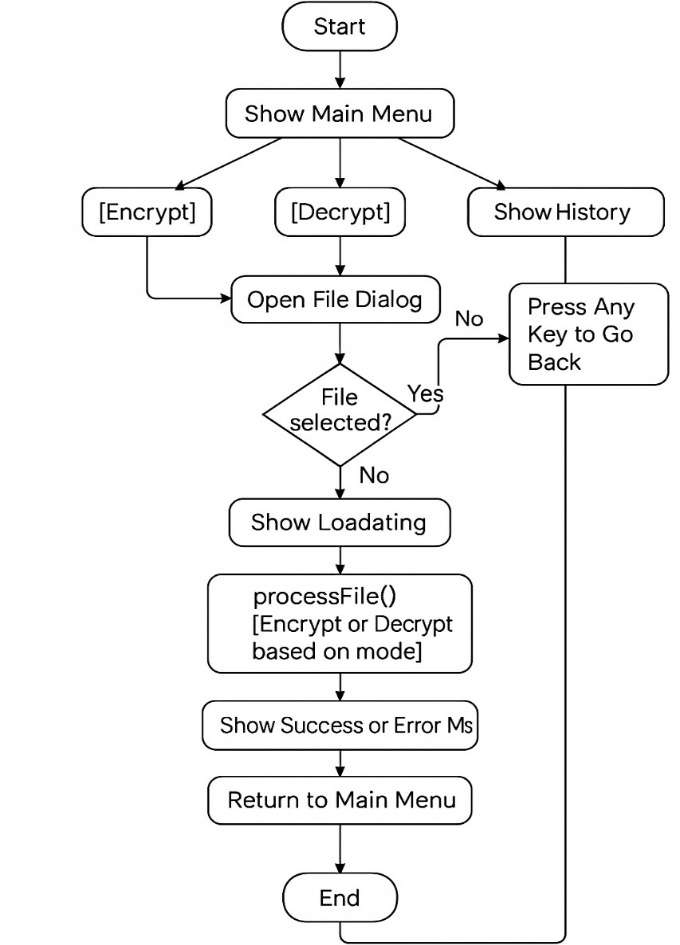


Fig 6: Activity diagram of file encryptor in c++

## Sequence Diagram

The sequence diagram of file encryption in C++ illustrates the interactions between various components involved in the encryption process over time. The key participants typically include the User, the Main Function, a File Handler (responsible for file I/O), and an Encryption Module (responsible for processing the encryption). The process begins when the User initiates the program. The Main Function first calls the File Handler to open and read the contents of the specified input file. Once the file is successfully read, the File Handler returns the raw content to the Main Function. The Main Function then passes this content along with an encryption key to the Encryption Module, which applies the encryption algorithm (such as XOR, Caesar Cipher, or AES) to the data. The encrypted content is returned to the Main Function, which then instructs the File Handler to save the encrypted data to a specified output file. Upon successful writing, a confirmation message is sent back to the User, indicating that the encryption process has been completed successfully. This sequence diagram clearly models the chronological flow of messages between the components, ensuring each step from input to encryption to output is carried out in a coordinated manner.

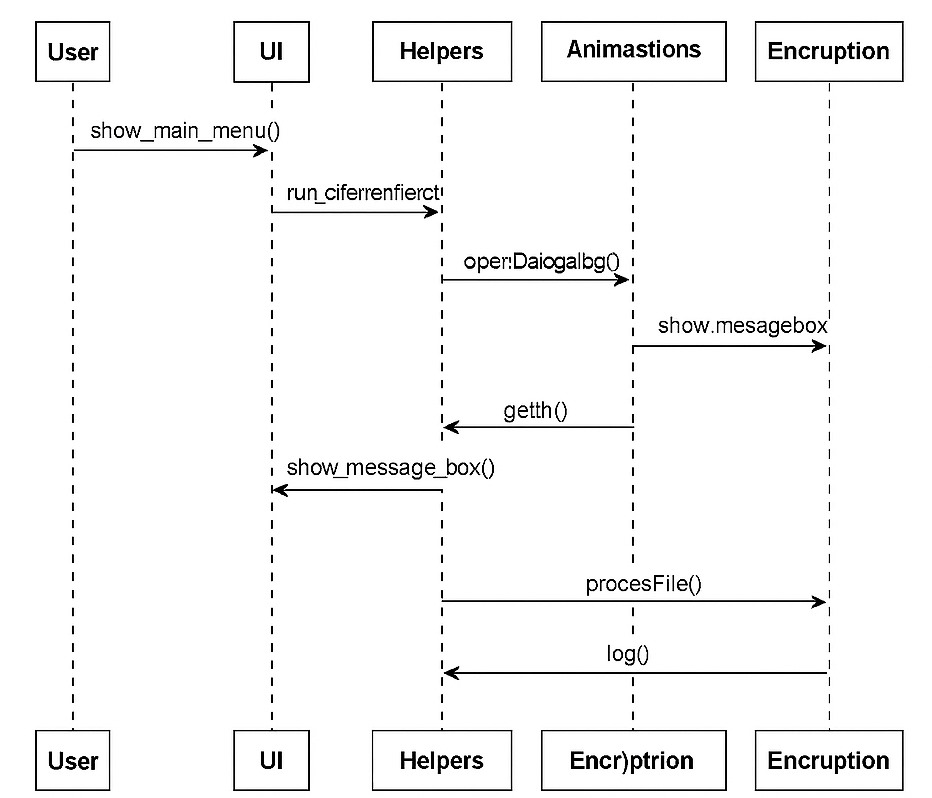


Fig 7: Sequence Diagram of file encryptor in c++

# System Development and Implementation

## Programming Platform

In the File Encryptor project using the Caesar Cipher algorithm in C++, a simple and user-friendly programming platform was used. The code was written in **C++,** a powerful and widely used programming language. For development, **VS code, DEV C** IDE was used, which provides a clean interface and easy-to-use features for writing, debugging, and running C++ programs. The **MSVC, GCC Compiler** was used to compile the C++ code. The project was created and tested on a **Windows operating system**, which made it easy to handle file operations. Basic text files (.txt) were used as input and output for encryption and decryption. The project did not require any external libraries, making it simple and lightweight. Overall, the platform was easy to set up and perfect for a beginner-level cryptography project.

## Operating Environment

The file encryptor project using the Caesar Cipher algorithm in C++ needs a basic computer system with a modern operating system like Windows, Linux, or mac OS. It requires a C++ compiler such as GCC (for Linux) or Min GW/MSVC (for Windows) to compile and run the code. The project can be developed using any simple code editor like Notepad++, VS Code, or Code::Blocks. It does not need high-end hardware; a computer with at least 2 GB RAM and a dual-core processor is enough. The system should have permission to read from and write to files on the local disk. No special software libraries are needed only the standard C++ libraries. The testing was done on basic text files to check if encryption and decryption worked properly.

# Assignment of Roles and Responsibilities

|  |  |  |
| --- | --- | --- |
| Member Name | Role & Responsibilities |  |
| Saraswati Rokaya |  |  |
| Salim Shrestha |  |  |
| Aayush Kumar Mallik |  |  |

Fig 9:Table of Role & Responsibilities

# Testing and Debugging

# Conclusion

In this project, we successfully created a file encryption system using the Caesar Cipher algorithm in C++. The Caesar Cipher is one of the simplest types of encryption, where each letter in the text is shifted by a fixed number of positions in the alphabet. Although it's a basic method, it helped us understand how encryption works and how data can be protected by making it unreadable to others.

By reading a file, encrypting its content using a key (shift value), and saving the encrypted output to another file, we learned how to handle file input/output and apply simple encryption logic in C++. This project gave us a good starting point for understanding the basics of cryptography and data security in programming.

# References