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

1. 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.

1. 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.

2. 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:

I. **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.

II. 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.

III. 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.

IV. 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.

V. 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.

VI. 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).

2.2 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)

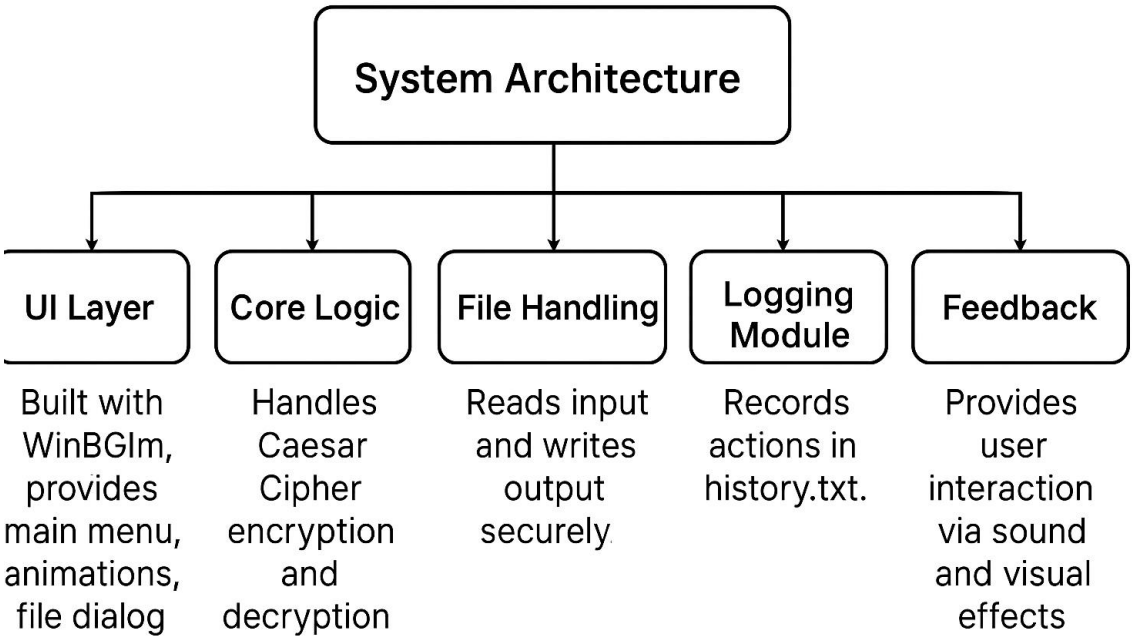
Fig 2: Table of file encryptor in c++

3. 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.



System Architecture

Flowchart

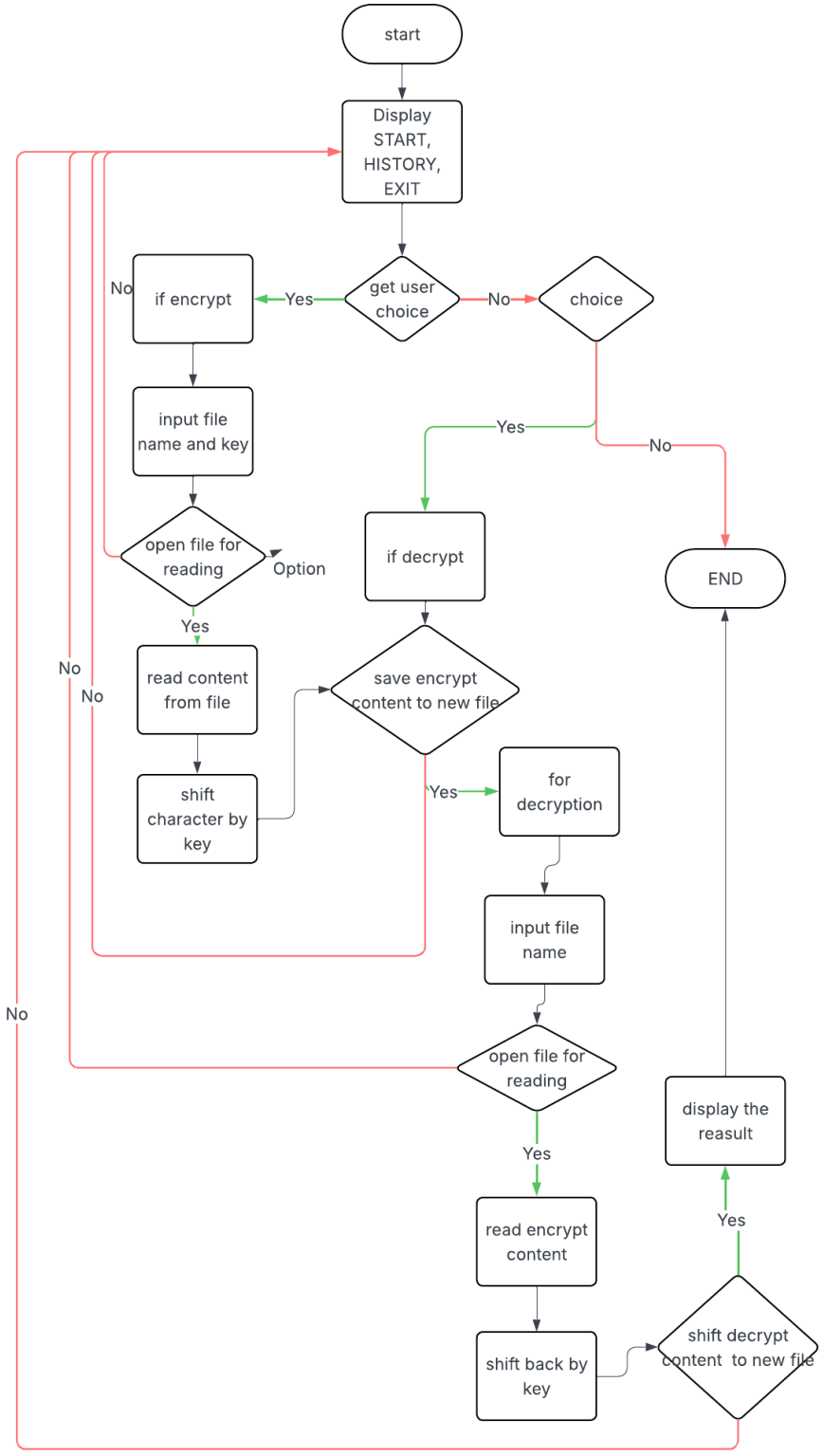


Fig 3: [Flowchart of file encryptor in c++]

4. 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.

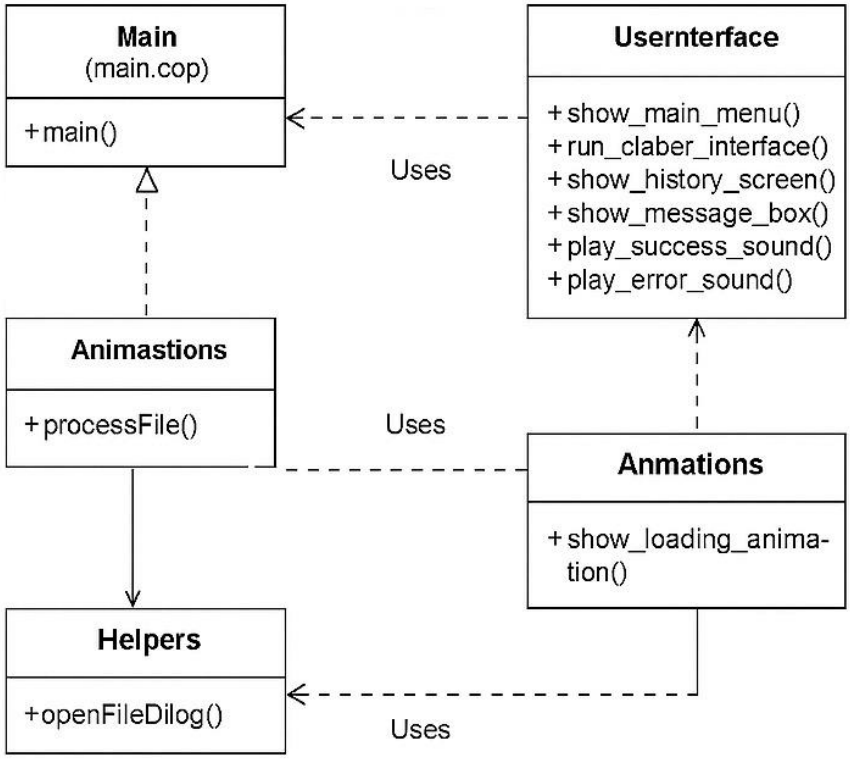


Fig 4: class diagram of file encryptor in c++

Use Case Diagram

Encryption Tool:
A C++ desktop application that allows users to encrypt and decrypt files with additional user-interface feedback features.

Primary Actor

- **User** – Interacts with the Encryption Tool through the graphical menu interface.

Use Cases and Descriptions

Use Case	Description
Select File	The user chooses a file from their system to be processed.
Encrypt File	The system applies a Caesar cipher to the selected file to produce an encrypted output.
Decrypt File	The system reverses the encryption on a selected file to restore the original content.
View History	The user can view a log of previously performed encryption and decryption actions.
Play Sound Feedback	The system plays success or error sounds to give immediate feedback after an operation.
See Loading Animation	The system shows a simple loading animation while processing to indicate progress.

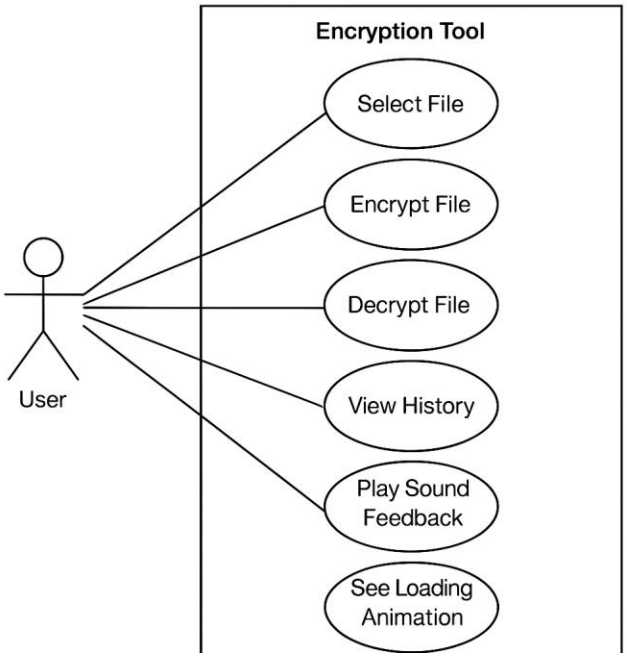


Fig 5: use case diagram of file encryptor in c++

Activity Diagram

The activity diagram illustrates the workflow of the **Encryption Tool** from start to end. It shows how the user navigates through the main menu and performs encryption, decryption, or views history.

Description of Activities

1. **Start**
   The application is launched and begins execution.
2. **Show Main Menu**
   The system displays a menu with three options: **Encrypt**, **Decrypt**, or **Show History**.
3. **User Chooses an Option**
   - **Encrypt** → proceed to select a file for encryption.
   - **Decrypt** → proceed to select a file for decryption.
   - **Show History** → directly displays the history log.
4. **Open File Dialog** *(for Encrypt or Decrypt only)*
   The system opens a file selection dialog.
5. **File Selected?**
   - **No** → displays a message prompting the user to press any key to return to the main menu.
   - **Yes** → continue to the next step.
6. **Show Loading Animation**
   While processing, the system displays a loading animation to indicate progress.
7. **Process File**
   The system applies the Caesar cipher:
   - If Encrypt mode: encrypts the file.
   - If Decrypt mode: decrypts the file.
8. **Show Success or Error Message**
   After processing, the system provides feedback (e.g., “Operation Successful” or an error message).
9. **Return to Main Menu**
   The system returns to the main menu for further actions.
10. **End**
    The process ends when the user chooses to exit the application.

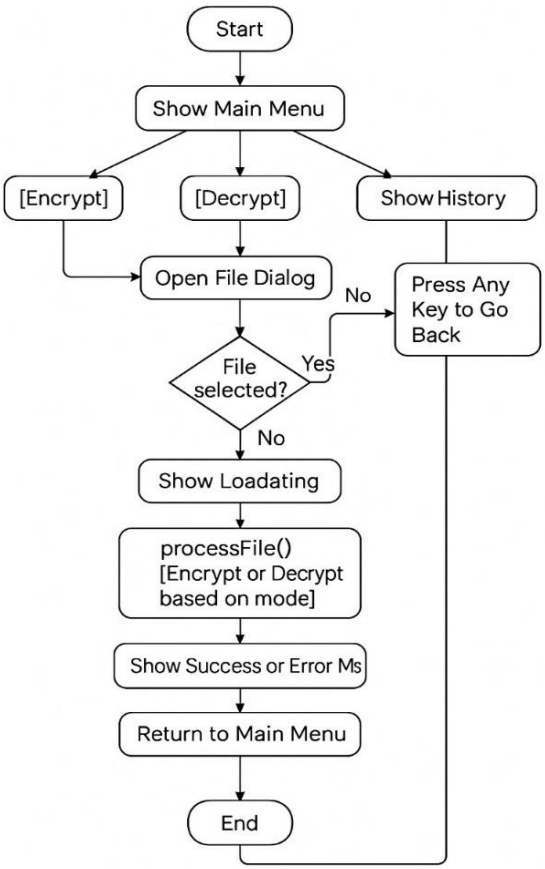


Fig 6: Activity diagram of file encryptor in c++

Sequence Diagram

The sequence diagram illustrates the interaction between the **User**, the **UI module**, and the internal components (**Helpers**, **Animations**, and **Encryption**) when performing an encryption or decryption operation.

Description of Interactions

1. **User → UI: show_main_menu()**
   The process starts when the user interacts with the application interface.
   The system displays the main menu and waits for the user's choice.

2. **UI → UI: run_cipher_interface()**
   When the user chooses to encrypt or decrypt, the UI module launches the cipher interface for further input.

3. **UI → Helpers: openFileDialog()**
   The UI calls the Helpers component to open a standard file dialog, allowing the user to select a file.

4. **Helpers → UI: (file path)**
   The Helpers component returns the chosen file path back to the UI.

5. **UI → Animations: show_messagebox()**
   Before processing, the UI triggers the Animations module to show a loading or processing message on the screen.

6. **UI → Encryption: processFile()**
   The UI then calls the Encryption component, passing the selected file and mode (Encrypt or Decrypt).
   The Encryption module applies the Caesar cipher algorithm to the file contents.

7. **Encryption → Encryption: log()**
   After processing, the Encryption module records the action (Encrypt or Decrypt) with the filename into a history log.

8. **Encryption → UI: (status)**
   The Encryption module returns the status of the operation (success or failure) to the UI.

9. **UI → User: show_message_box()**
   Finally, the UI displays a success or error message back to the user.

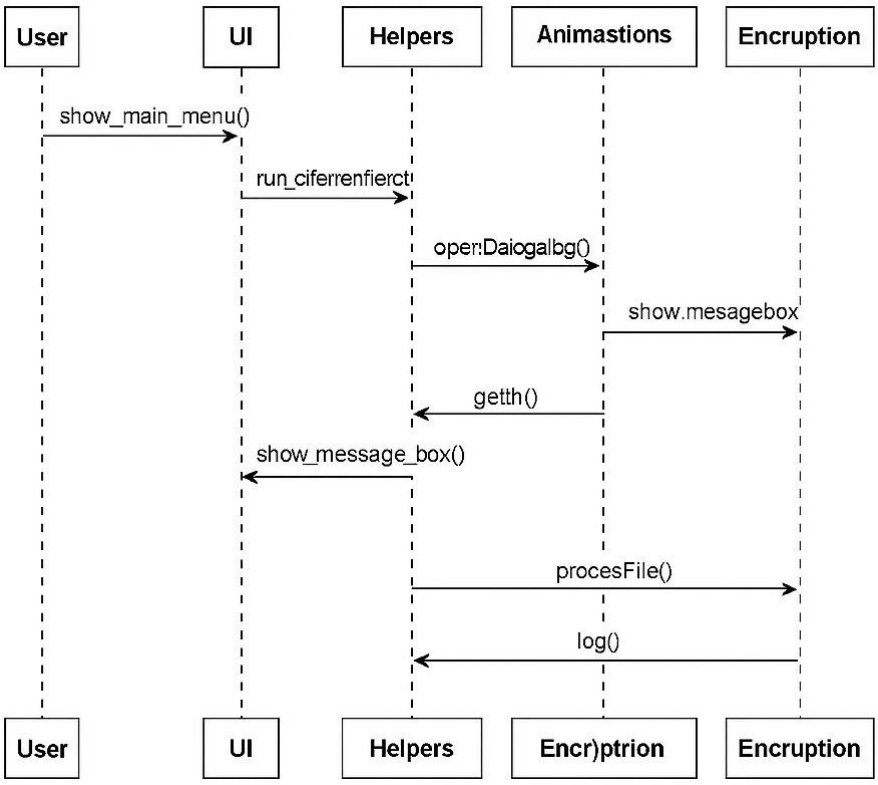


Fig 7: Sequence Diagram of file encryptor in c++

5. System Development and Implementation

Programming Platform

The **File Encryptor** project, implemented using the Caesar Cipher algorithm in C++, was developed on a simple yet efficient programming setup.

C++ was chosen for its performance, portability, and extensive standard library support. The development environment included **Visual Studio Code** and **Dev-C++ IDE**, both of which provide a clean interface, syntax highlighting, and debugging features that make development easier and more organized.

For compilation, standard compilers such as **MSVC** (Microsoft Visual C++) and **GCC/MinGW** were used. These compilers ensure that the code runs efficiently on Windows systems. The project was designed and tested primarily on a **Windows operating system**, making use of its straightforward file handling and graphics support.

The tool works with basic text files (.txt) as input and output for encryption and decryption, which makes testing and usage simple.

No third-party libraries were required—only standard C++ libraries—keeping the project lightweight and beginner-friendly.

Operating Environment

The **File Encryptor** is lightweight and can run on any modern computer with a basic setup. It is compatible with popular operating systems such as **Windows**, **Linux**, or **macOS**, provided that a C++ compiler (e.g., **GCC**, **MinGW**, or **MSVC**) is installed.

Development and usage do not demand high-end hardware. A system with at least:

- **2 GB RAM**,
- a **dual-core processor**, and
- sufficient disk space to store input and output files

is adequate to run the application smoothly.
The tool can be built and executed using common code editors or IDEs like **Visual Studio Code**, **Dev-C++**, **Code::Blocks**, or even **Notepad++** for quick edits.

The only requirement is that the system must have permission to read from and write to the local disk where the input and output files are stored.

All testing was performed using standard text files to ensure that both encryption and decryption functions work correctly. Because no special libraries or frameworks are required, the project is easy to set up, portable, and ideal for learning or demonstration purposes.

6. Assignment of Roles and Responsibilities

Member Name	Role & Responsibilities
Saraswati Rokaya	**UI & User Interaction:** Designed the user interface (menus, buttons, messages), implemented graphical feedback (loading animation, message boxes), and integrated sound feedback for user actions.
Salim Shrestha	**Core Logic & Integration:** Implemented the main encryption/decryption logic using the Caesar Cipher, managed file input/output operations, and integrated all modules into a working system.
Aayush Kumar Mallik	**Testing & Documentation:** Performed testing with different files to ensure correct functionality, debugged issues, maintained history log features, and prepared project documentation including diagrams and proposal sections.

Fig 9: Table of Role & Responsibilities

7. Testing and Debugging

Testing Approach

The **File Encryptor** project was thoroughly tested to ensure that all core features work as expected. Testing was carried out using both valid and invalid input files to cover different scenarios.

The main objectives of testing were:

- To verify that encryption and decryption produce correct results.
- To confirm that the user interface responds properly to all interactions.
- To ensure that the history log, sound feedback, and loading animation work smoothly without crashing.

Test Cases:

Test Case	Description	Expected Result	Outcome
Encrypt valid file	Select a .txt file and encrypt it	Encrypted file (encrypted_filename.txt) is created	Pass
Decrypt valid file	Select an encrypted file and decrypt it	Original content restored in decrypted_filename.txt	Pass
View history	Open history log after several operations	All actions are listed with timestamps	Pass
Cancel file selection	Close file dialog without selecting a file	Show message and return to main menu	Pass
Missing file	Select a non-existing file	Error message displayed, no crash	Pass

Debugging Process

During development, debugging was performed iteratively:

1. **Compile-time Errors:**
   Fixed syntax errors and type mismatches during compilation using IDE error messages and compiler feedback.
2. **Runtime Testing:**
   - Ran the program step by step to identify logic issues.
   - Checked edge cases like empty files or special characters.
   - Verified that file handles were properly closed after operations.
3. **UI and Graphics Debugging:**
   - Adjusted button coordinates and click regions.
   - Tested sound file paths and confirmed animations did not overlap text.
4. **Logging and Verification:**
   - Used history.txt entries to cross-check whether encryption/decryption actions were recorded accurately.
   - Re-decrypted files to confirm content integrity.

8. Conclusion

In this project, we successfully developed a file encryption system using the Caesar Cipher algorithm in C++. The Caesar Cipher, one of the simplest forms of encryption, involves shifting each letter in the text by a fixed number of positions in the alphabet. While the method itself is basic and not suitable for advanced security applications, it served as an effective way to explore the core principles of encryption.

Through the process of reading file content, applying the Caesar Cipher transformation, and writing the encrypted or decrypted output to a new file, we gained practical experience in file handling, user interaction, and implementing logical operations in C++. Additionally, we integrated a graphical user interface using the WinBGIm library, along with sound and visual feedback, to enhance user experience.

Overall, this project provided us with a solid foundation in understanding how cryptographic systems function and how security features can be integrated into real-world applications. It has also prepared us to explore more complex encryption techniques and data protection strategies in future projects.

9. References

1. TutorialsPoint.
    - Caesar Cipher Algorithm in C++
2. WinBGIm Documentation.
    - WinBGIm graphics library
3. GeeksforGeeks.
    - File Handling in C++
4. GitHub Repositories and Community Forums for code structure and implementation tips.