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> <73176>

<Aayush Kumar Mallik> <731744>

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

2025

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 > <73176>

< Aayush Kumar Mallik > < 731744>

**Project Supervisor**

**Mr. Kiran Khanal**

**Senior Assistant Professor**

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

2025

**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. 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 user can choose the shift key (also called offset) to control how much each letter is moved. 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. A simple command-line interface makes it easy for users to enter file names and key values. 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 would like to express heartfelt gratitude to all those who supported us throughout the development of this project titled **“File Encryptor in C++”**. First and foremost, We would like to sincerely thank our project supervisor, **Mr. Kiran Khanal**, for his invaluable guidance, encouragement, and constructive feedback during every stage of this project. His deep knowledge and constant motivation were instrumental in shaping this work. We also thankful to the faculty members and staff of the **Department of Computer Engineering** for providing a supportive academic environment and the necessary infrastructure to complete this project. Our sincere appreciation also goes to my friends and classmates who provided insightful suggestions and encouragement throughout the development process. Last but not least, we would like to thank my family for their unwavering support, understanding, and patience during the course of this project.

# 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 Stamp:

Er. Subash Rajkarnikar

HoD

Department Of Computer Engineering,

Kantipur City College

Date:- 2025

# Table Of Content

# List Of Figures

Fig 1: Waterfall Model

Fig 2: Table of file encryptor in c++

Fig 3: Flowchart of file encryptor in c++

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

Fig 5: Activity diagram of file encryptor in c++

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

Fig 7: **ER (Entity-Relationship)** diagram of file encryptor in c++

Fig 8: Table of Role & Responsibilities

# Introduction

## 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.

Developed using C++, this project focuses on simplicity, speed, and effectiveness. It allows users to encrypt and decrypt various types of files through a user-friendly command-line interface. The project implements both basic XOR encryption for learning purposes and more secure methods such as AES (Advanced Encryption Standard) for practical use. The File Encryptor not only enhances understanding of file handling and cryptography but also promotes awareness of digital security in real-world applications.

## Overview

The File Encryptor project is a desktop-based application developed in C++ that enables users to securely encrypt and decrypt files to protect sensitive data from unauthorized access. With growing concerns over data privacy and cyber security, this tool offers a practical solution for safeguarding personal or confidential information by applying encryption algorithms to file content. Users can select a file, choose an encryption method, and provide a key or password to convert readable data into an unreadable format (cipher text), which can only be restored to its original form using the correct key.

The application primarily supports symmetric encryption techniques, such as a basic XOR cipher and may be extended to implement more advanced methods like AES. It includes features like file validation, key verification, and error handling to ensure a smooth and secure user experience. Designed with simplicity and efficiency in mind, this project demonstrates how encryption works at a fundamental level and provides a solid foundation for students to explore real-world applications of cryptography and data protection.

## 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

1. Authentication

* Ensures integrity and authenticity using GCM or HMAC.

1. Initialization Vector

* Random IV for each encryption session to ensure randomness (especially for AES in CBC/CTR mode).

1. Stream Encryption

* Encrypt/decrypt file in chunks to avoid loading large files into memory.

## Objective

1. Confidentiality:

* Ensure that the contents of a file cannot be understood by unauthorized users.
* Encryption transforms readable data (plaintext) into unreadable data (ciphertext).

1. Data Integrity:

* Prevent undetected modification of the file during storage or transmission.
* While encryption alone doesn’t guarantee integrity, it's often paired with checksums or hashes for this purpose.

1. **Access Control:**

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

1. **Secure Storage:**

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

1. **Secure Transmission:**

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

## Significance

A file encryptor in C++ is a program that protects the contents of a file by turning it into unreadable code using encryption techniques. This is important because it keeps sensitive information like passwords, personal data, or private documents safe from hackers or unauthorized users. Only someone with the correct key or password can unlock (decrypt) the file and read its original content.

Using C++ for file encryption is powerful because it gives full control over how files are handled and how secure the encryption process is. C++ also runs very fast, making it a good choice for encrypting large files or building secure systems like password managers, secure backups, or confidential file transfer tools.

# Scope and Limitation

## Scope

This project aims to develop a simple file encryptor using C++ that allows users to securely encrypt and decrypt files to protect their data from unauthorized access. It will use a basic encryption algorithm (like AES) to convert the contents of a file into unreadable text and then restore it using a correct password or key. The program will support text or binary files and allow users to choose files from their system through a command-line interface. It will be designed to work on most operating systems and will focus on protecting personal or sensitive data in a lightweight and user-friendly way.

## 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.

 No user interface only works through command-line input.

 ️ Does not provide protection against advanced attacks like side-channel or brute-force unless combined with strong passwords.

# Methodology

## Software Development Life cycle

**Waterfall Model**

****

Fig 1: Waterfall Model

1. Requirement Analysis

* Define what the program should do: encrypt/decrypt a file using Caesar Cipher, take input/output file names, and shift key.
* Decide on supported file formats (e.g., .txt only).
* Define expected behavior for invalid inputs or characters.

1. System Design

* Design the structure of the program (functions for reading, encrypting, writing).
* Plan file handling, user input, and error checking.
* Decide how Caesar Cipher will be implemented (ASCII character shifting, handling upper/lowercase).

1. Implementation

* Write the program in C++.
* Create functions for:
* Reading input from a file
* Encrypting text using Caesar Cipher
* Writing to the output file

1. Testing

* Test with various shift keys and input files.
* Check encryption correctness and file handling robustness.
* Validate handling of special characters, empty files, etc.

1. Deployment

* Deliver the final executable or source code.
* Document usage instructions for the user.

1. Maintenance

* Fix any bugs found after deployment.
* Optional: Add new features like decryption, GUI, or support for other ciphers.

# Technologies and Tools

## 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 architecture of the **File Encryptor in C++** is designed to be simple, modular, and easy to follow. The program starts with a **user input module**, where the user provides the path of the file to be encrypted or decrypted, chooses the operation type (encrypt or decrypt), and enters the **key value**, which is an integer used in the Caesar Cipher to shift characters. After collecting the input, the system moves to the **file handling module**, which opens the selected file in read mode and prepares to process its contents. The file can be either a text or binary file, but in this Caesar Cipher implementation, it's mainly used for text-based files for clarity. Once the file is opened, the content is passed to the **encryption/decryption module**, which applies the Caesar Cipher logic. For encryption, each character in the text is shifted forward by the key value in the ASCII table; for decryption, each character is shifted backward. Special care is taken to handle uppercase, lowercase letters, and non-alphabetic characters appropriately so that the output remains readable and accurate.

After the content is encrypted or decrypted, the data is passed to the **output module**, which creates a new file and writes the processed content to it. This ensures that the original file remains unchanged, and a separate file is generated for secure storage or sharing. The architecture also includes an **error-handling system** that checks for missing files, invalid key values, or unreadable characters and gives appropriate error messages. All operations are performed in a step-by-step manner to keep the system lightweight and fast. The architecture is linear and follows a clear flow from input to processing and then output. It is built using C++ features like file streams, loops, and conditionals to handle logic and file operations efficiently. The modular structure makes it easy to understand, test, and modify. Additional features such as menu options, logs, and integrity checks can be added without disrupting the core structure. Overall, this system architecture provides a clean and practical framework for learning basic file encryption using one of the simplest algorithms in cryptography the Caesar Cipher.

## Flowchart

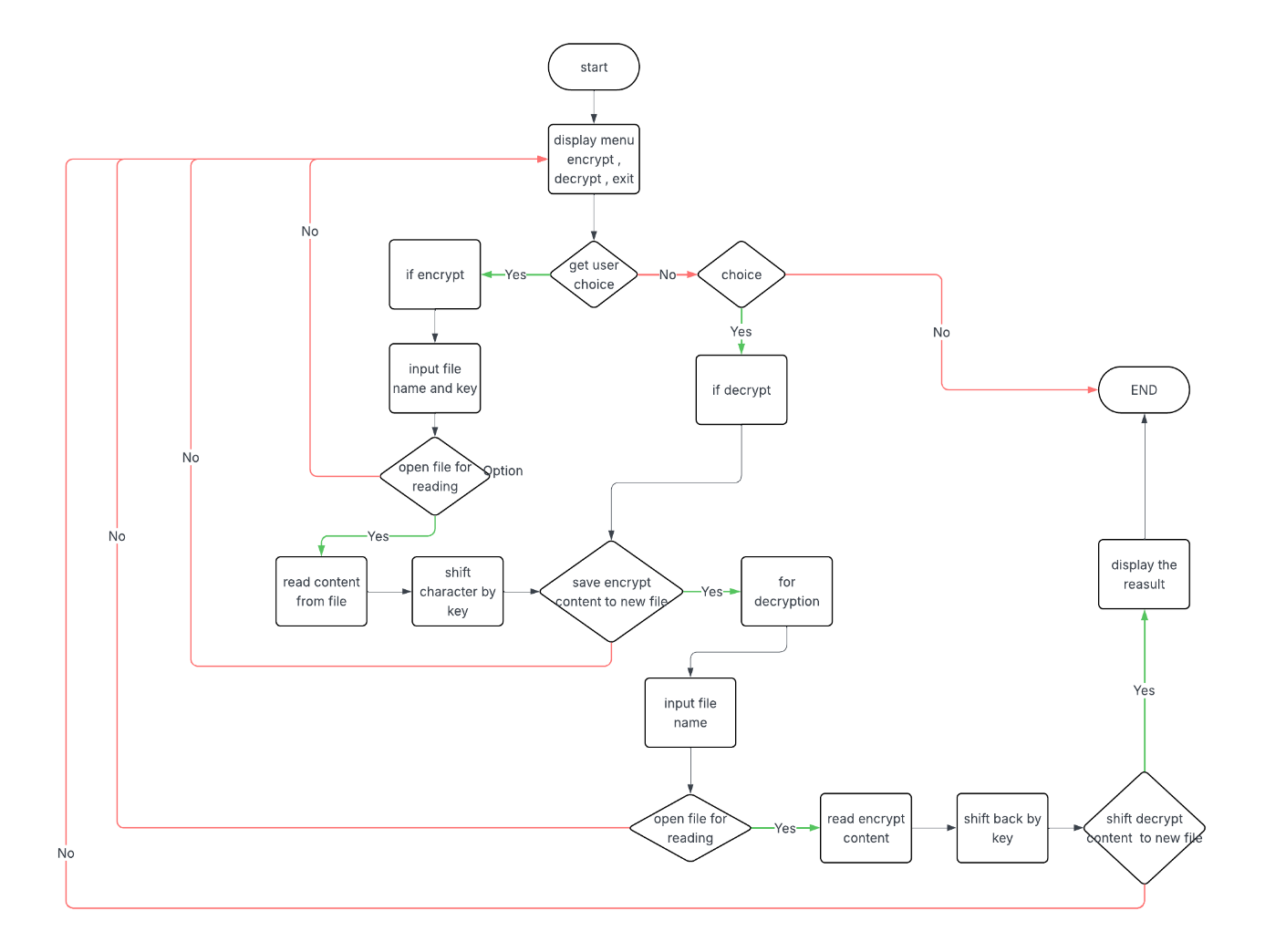


Fig 3: Flowchart of file encryptor in c++

# 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.

## 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.

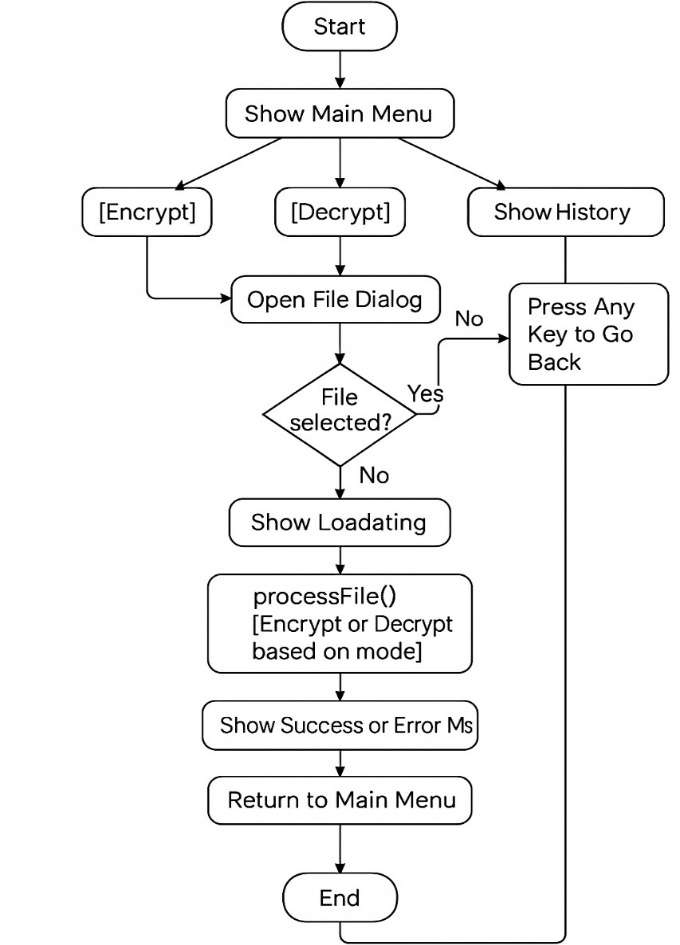


Fig 5: Activity diagram of file encryptor in c++

## 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.

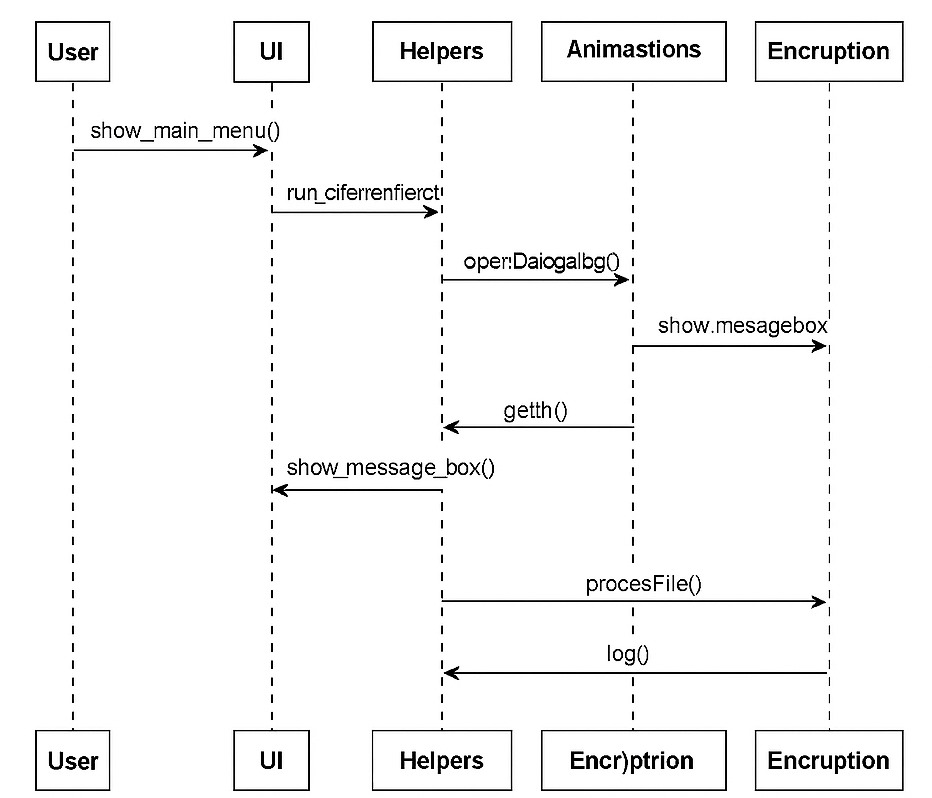


Fig 6: Sequence Diagram of file encryptor in c++

# Database Design

## ER Diagram

An **ER (Entity-Relationship) diagram** for a file encryptor in C++ using the **Caesar Cipher algorithm** shows how different parts of the program interact. The main entity is the **User**, who interacts with the system to encrypt or decrypt files. The **File** entity holds attributes like filename, file path, and content. A **Caesar Cipher** entity performs the encryption and decryption, using a **Key** (shift value) attribute. The **Operation** entity stores details about the action (encrypt or decrypt) and the timestamp. There's also a **Log** entity that records user activity for reference. Relationships exist between User and Operation, File and Operation, and Caesar Cipher and Operation, showing how each component works together. This diagram helps visualize the logical flow and data structure of the encryption system in a simple way.

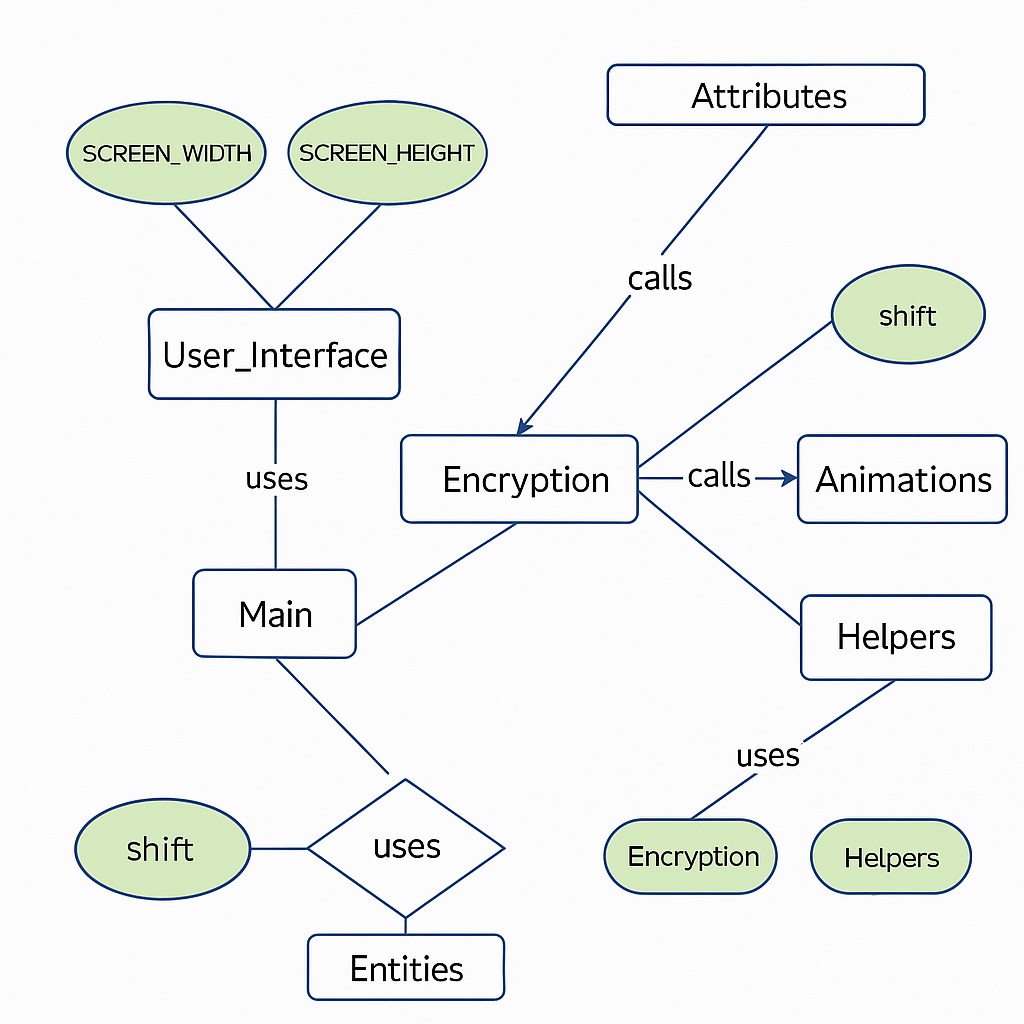


Fig 7: **ER (Entity-Relationship)** diagram of file encryptor in c++

# 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 8: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