# PASSWORD MANAGER

**A PROJECT REPORT**

***Submitted by***

**Ishan-23BCS12656**

**Soumyah-23BCS12930**

***In partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

COMPUTER SCIENCE AND ENGINEERING

**Chandigarh University**

JUNE 2025

# BONAFIDE CERTIFICATE

Certified that this project report **" PASSWORD MANAGER"** is the bonafide work of "ISHAN & SOUMYAH**"** who carried out the project work under my/our supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **DR. Sandeep Singh Kang** | **Er. Shefali Goyal** |
| **HEAD OF THE DEPARTMENT**<br>**BE- Computer Science and Engineering**<br>**Third Year Engineering** | **(SUPERVISOR)** |

Submitted for the project viva-voice examination held on-24/06/2025

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# TABLE OF CONTENTS

## List of Figures

# List of Tables

# INTRODUCTION

## INTRODUCTION

## 1.1 Identification of Client / Need / Relevant Contemporary Issue

In the digital age, managing passwords securely has become a critical necessity. With the average internet user having over 100 online accounts, remembering and maintaining strong, unique passwords is both challenging and essential. Weak or reused passwords significantly increase the risk of data breaches, identity theft, and unauthorized access.

Traditional password managers are either too complex or paid, leaving a gap for simple, educational, and secure systems that demonstrate encryption and hashing techniques.

This project addresses the modern-day need for a user-friendly password management system that incorporates basic encryption, hashing, and prioritization based on password strength, which can also serve as an academic demonstration of key cybersecurity principles.

## 1.2 Identification of Problem

The main problem is the lack of accessible and easy-to-understand tools that demonstrate how encryption and hashing techniques work together in securing passwords. Key issues include:

- Users often use weak or repeated passwords.
- Lack of awareness about password strength metrics.
- Inaccessible or overly complex password tools for learners or basic users.

## 1.3 Identification of Tasks

To overcome the above challenges, the following tasks were defined:

- Develop a simple C++ based password manager.
- Implement Caesar Cipher encryption to simulate data obfuscation.
- Use hashing (std::hash) to simulate secure storage.
- Calculate and score password strength.
- Store and prioritize passwords using a priority queue based on their strength.
- Design a CLI interface for user interaction.

## 1.4 Timeline

| Tasks | Week 1 | Week 2 | Week 3 | Week 4 |
|---|---|---|---|---|
| Requirement analysis & planning | ✓ | | | |
| Caesar Cipher & hashing implementation | | ✓ | | |
| Password strength logic & priority queue | | ✓ | ✓ | |
| Documentation & final report preparation | | | | ✓ |

## 1.5 Organization of the Report

This report is structured into five chapters:

- **Chapter 1: Introduction** – Discusses the problem, motivation, and project objectives.
- **Chapter 2: Literature Review / Background Study** – Reviews existing password handling systems, their strengths and weaknesses, and academic research.
- **Chapter 3: Design Flow / Process** – Describes the architecture, design decisions, encryption and hashing strategy, and implementation steps.
- **Chapter 4: Results Analysis and Validation** – Analyzes how well the password manager performs in terms of features and security.
- **Chapter 5: Conclusion and Future Work** – Summarizes the work done and outlines possible improvements and extensions.

# CHAPTER 2. LITERATURE REVIEW / BACKGROUND STUDY

## 2.1 Timeline of the Reported Problem

In recent years, cybersecurity threats have increased dramatically due to the widespread use of online platforms and digital identities. Passwords remain the primary method for user authentication across the internet, yet many users continue to reuse weak or easily guessable passwords. This ongoing problem has prompted a growing focus on password management tools and techniques in academic and industry research.

## 2.2 Existing Solutions

There are several commercial and open-source password managers, such as LastPass, 1Password, Bitwarden, and KeePass. These tools offer features like encrypted storage, password generation, and synchronization across devices. However, many:

- Rely on complex encryption methods that are not easy for beginners to understand.
- Require user registration, internet connectivity, or subscription plans.
- Do not transparently demonstrate how password strength or hashing mechanisms work under the hood.

These tools are practical but lack educational value, especially for students trying to learn basic encryption and password handling mechanisms.

## 2.3 Bibliometric Analysis

Academic studies over the last decade highlight increasing interest in password security mechanisms. Common themes include:

- Encryption algorithms like AES, RSA, and simple ciphers for educational use.
- Hashing techniques such as MD5, SHA-256, and their role in secure password storage.
- Password strength metrics, evaluating the effectiveness of user-selected passwords.
- Use of data structures like priority queues or heaps to organize and prioritize password-related data.

These studies suggest a growing demand for simple yet effective implementations for learning and demonstration purposes.

## 2.4 Problem Definition

Most password managers focus on functionality and security but ignore educational clarity. For students learning about cybersecurity, there is a need for a minimal, easy-to-understand system that demonstrates:

- Basic encryption using Caesar Cipher.
- Hashing using built-in C++ functions.
- Password strength evaluation logic.
- Efficient storage and retrieval using data structures like priority queues.

### 2.5 Review Summary

**3**

| Aspects | Strengths of Existing Solutions | Limitations |
|---|---|---|
| Encryption | Strong security with AES, bcrypt, etc. | Complex to implement or understand for beginners |
| Hashing | Secure and widely used algorithms | No visibility of internal logic in commercial tools |
| Usability | Cross-platform, cloud sync features | Subscription required; overkill for simple needs |
| Educational Value | Limited in commercial tools | Few tools demonstrate basics like Caesar Cipher or std::hash |

### 2.6 Goals / Objectives

- Design a **console-based password manager** using C++.
- Implement **Caesar Cipher encryption** for basic obfuscation.
- Simulate password hashing using `std::hash`.
- Score password strength using simple metrics (length, digits, symbols).
- Prioritize and organize passwords based on strength using a **max-heap** (priority queue).
- Provide a simple, interactive CLI for users to store and view passwords securely.

# CHAPTER 3. DESIGN FLOW / PROCESS

## 3.1 Evaluation & Selection of Specifications / Features

To fulfill the educational and functional goals of the password manager, the following key specifications and features were selected:

- **Caesar Cipher encryption** to demonstrate a simple, character-based encryption method.
- **C++ Standard Library's std::hash** for password hashing simulation.
- **Password strength evaluation logic** based on length, digit presence, and symbols.
- **Priority Queue (Max-Heap)** to organize stored passwords based on their strength.
- **Console-based User Interface (CLI)** for simplicity and portability.

These features were chosen to balance security principles with ease of implementation and educational value.

## 3.2 Design Constraints

While designing the system, several constraints were considered:

- **Limited encryption complexity**: For educational clarity, Caesar Cipher (which is not secure in real-world use) was used intentionally.
- **No external libraries**: The entire system was implemented using standard C++ libraries.
- **Console-only interface**: To keep the focus on core logic and minimize dependency on UI frameworks.
- **Storage in memory**: Passwords are not saved to disk, meaning the data resets on each run (simplifying implementation).

## 3.3 Analysis of Features and Finalization Subject to Constraints

The following design trade-offs were finalized:

- Chose Caesar Cipher over AES to maintain simplicity and show how letter shifting works.
- Used std::hash<string> as a stand-in for real cryptographic hash functions (e.g., SHA-256).
- Used a priority queue to allow dynamic sorting of passwords by strength rather than storing in arrays or lists.
- Implemented a score-based strength system (0–3) to keep evaluations lightweight and understandable.

## 3.4 Design Flow

- **Frontend**:
  Using HTML, CSS and Java Script.

- **Backend**:
  Core logic implemented in C++ for encryption, hashing, strength evaluation, and password management.

- **Data Storage**:
  Passwords are stored temporarily in memory using vector and priority queue.

- **Integration**:
  Combines Caesar Cipher for encryption, std::hash for hashing, and a priority queue to sort passwords by strength.

## 3.5 Design Selection

The following tools and libraries were selected for implementation:

- **Language**: C++
- **Libraries**: <iostream>, <string>, <vector>, <queue>, <unordered_map>, <functional>, <cctype>
- **Data Structure**: std::priority_queue to sort and access passwords by strength.
- **Algorithm**: Caesar Cipher for encryption, std::hash for hashing, character analysis for scoring.

These choices allowed a balance between clarity, performance, and adherence to C++ standard practices.

## 3.6 Implementation Plan / Methodology

- **Phase 1: Requirement analysis and feature selection**
  Identified the need for encryption, hashing, password strength evaluation, and strength-based sorting using standard C++.

- **Phase 2: CLI design and interaction flow**
  Designed a simple, menu-driven console interface for user input and output.

- **Phase 3: Core logic integration**
  Implemented Caesar Cipher for encryption, std::hash for hashing, and a scoring system for evaluating password strength.

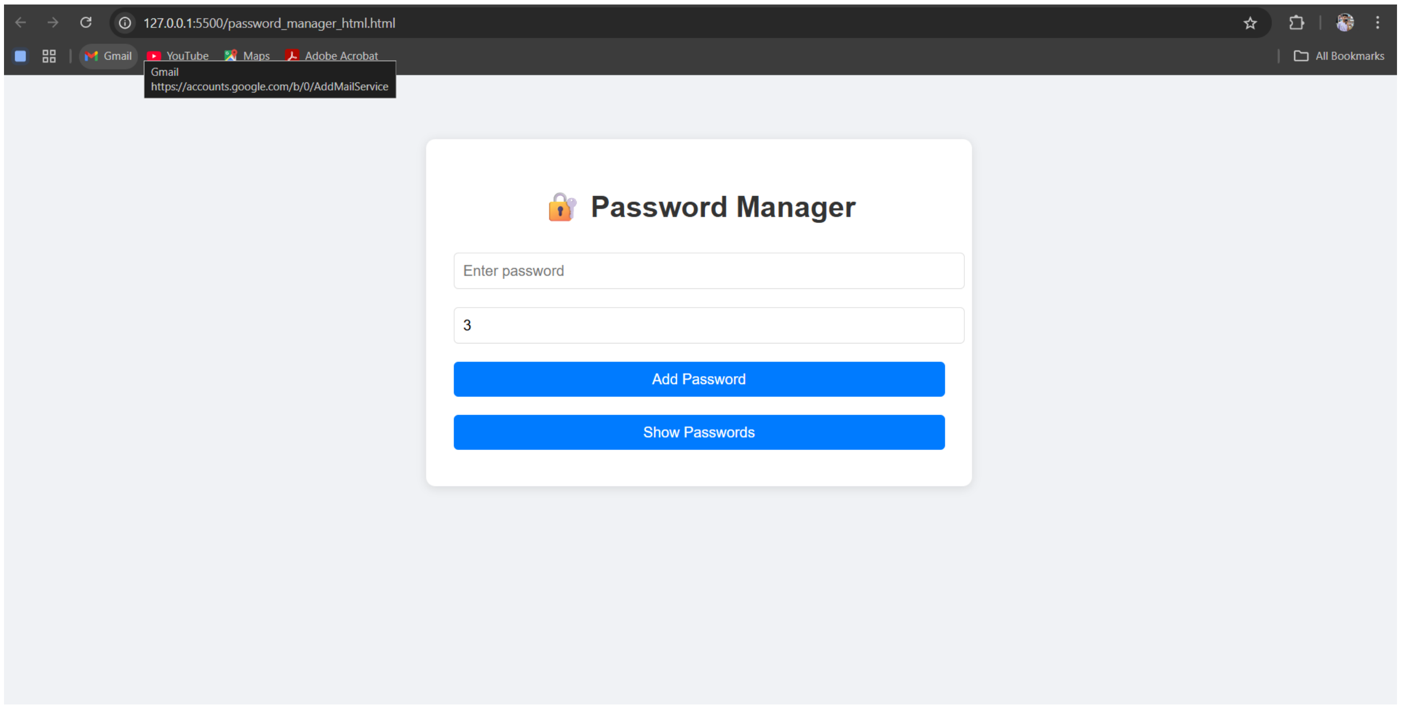
- **Phase 4: Priority queue integration and testing**
  Integrated a priority queue to organize passwords based on strength and tested the functionality with various inputs.

# CHAPTER 4. RESULTS ANALYSIS AND VALIDATION

## 4.1 Implementation of Solution

The Password Manager was successfully implemented in C++ as a console-based application. The solution integrates Caesar Cipher encryption, password hashing using std::hash, a strength evaluation system, and priority queue-based organization of passwords. The following components were tested and validated

## 4.2 ScreenShot



# CHAPTER 5. CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

The Password Manager project successfully demonstrates the fundamental principles of password handling, including encryption, hashing, and strength evaluation, using standard C++. By integrating Caesar Cipher for encryption and `std::hash` for hashing, the system provides a foundational understanding of how passwords can be securely managed.

The use of a priority queue enabled dynamic organization of password entries based on strength, ensuring that the most secure passwords are prioritized in display. The console-based interface kept the application simple, lightweight, and easy to use for learners and evaluators.

Overall, the project met its objectives by combining core computer science concepts — such as data structures, basic cryptography, and user input handling — into a functional, educational tool.

## 5.2 Future Work

To enhance functionality and practical utility, the following improvements are proposed for future development:

- **Persistent Storage**: Implement file-based or database storage to retain password data across sessions.
- **Password Generator**: Add a feature to create strong, randomized passwords automatically.
- **Enhanced Encryption**: Replace Caesar Cipher with more secure encryption algorithms like AES or bcrypt.
- **GUI Integration**: Develop a graphical user interface using libraries such as Qt or web technologies for improved user experience.
- **Export/Import Functionality**: Enable users to export encrypted password data and import it later for backup or transfer.

## USER MANUAL

- Installation and Setup

- Download or clone the project files.

- Open the file in your preferred C++ IDE or compiler.

- Compile the code using the appropriate build command.

  For example (in terminal): `g++ password_manager.cpp -o password_manager`

- Run the compiled program: `./password_manager`

### Troubleshooting

- **If the program crashes on input,** ensure you are entering valid data (e.g., non-empty password, numeric shift value).

- **If passwords are not retained after exit,** remember that this version uses in-memory storage only — data will be lost on closing the program.

- **If compilation errors occur,** verify that you are using a C++11 or newer compatible compiler (e.g., g++ with -std=c++11).

- **If encrypted output appears incorrect,** check the Caesar shift value you provided — it should be a valid positive integer.

## REFRENCES:-

- Bishop, M. (2003). *Introduction to Computer Security*. Addison-Wesley.
  → Concepts of password security, encryption, and hashing.

- Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7th Edition). Pearson Education.
  → Overview of Caesar Cipher and hashing mechanisms.

- GeeksforGeeks – https://www.geeksforgeeks.org/
  → C++ implementations of data structures like priority_queue, and basics of Caesar Cipher.

- cplusplus.com – https://cplusplus.com/reference/
  → Official documentation of C++ Standard Library functions like std::hash, std::queue, std::string.

- OWASP Foundation – https://owasp.org
  → Guidelines and best practices for password handling and security principles.

- ISO/IEC 27001:2013 – International Standard for Information Security Management
  → Referenced for general principles of secure data handling.