Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved; NAAC Accredited A++
Karunya Nagar, Coimbatore - 641 114, Tamil Nadu, India.

PassGuard: A PASSWORD ENCRYPTION TOOL FOR SECURE CREDENTIAL MANAGEMENT

a mini-project report submitted by

JOSHUA SIBICHAN SCARIYA– URK25CS1098

RONEL ABRAHAM MATHEW – URK25CS1106

ADITI LAKSHMANAN – URK25CS1113

YOGESH S – URK25CS1256

in partial fulfillment of the Continuous Internal Examination

of

25CS207- PROGRAMMING FOR PROBLEM SOLVING IN C

in

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Dr. R. Venkatesan, Associate Professor.

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

November 2025

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved; NAAC Accredited A++
Karunya Nagar, Coimbatore - 641 114, Tamil Nadu, India.

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this mini project report titled **"PassGuard: A PASSWORD ENCRYPTION TOOL FOR SECURE CREDENTIAL MANAGEMENT"** is the Bonafide work of

1. JOSHUA SIBICHAN SCARIYA– URK25CS1098
2. RONEL ABRAHAM MATHEW – URK25CS1106
3. ADITI LAKSHMANAN– URK25CS1113
4. YOGESH S – URK25CS1256

who carried out the mini project work under my supervision.

Dr. R. Venkatesan
Supervisor
Associate Professor
Division of Computer Science and Engineering
Karunya Institute of Technology & Sciences

Examiner

Submitted for the Project Viva Voce held on 22-11-25

ABSTRACT

This project tackles the critical challenge of securing user credentials in the digital age by developing a Password Encryption Tool using C. The main focus is on building core programming skills while introducing foundational cybersecurity concepts.

The application collects username and password pairs from users through a clear, menu-driven interface. To ensure safe and valid data entry, control statements and logical checks are used to validate inputs and prevent issues like buffer overflow. Once validated, passwords are transformed with string handling functions in preparation for encryption.

At its heart, the tool uses a simple Caesar cipher (character shifting) algorithm to encrypt passwords, keeping them safe from direct exposure in storage. All encrypted credentials are saved alongside usernames using C’s file handling features, ensuring persistence and data integrity with robust error handling and proper management of file pointers and access modes.

For authentication, the tool loads stored entries, re-encrypts the user’s login attempt, and compares it to the encrypted record using straightforward linear search techniques. This secure comparison ensures only correct credentials are accepted.

By combining essential C programming elements—such as loops, functions, pointers, and strings—with basic cryptography, the project not only delivers a practical password management solution but also lays the groundwork for further enhancements, such as more advanced encryption or network-based login features. Ultimately, this tool strengthens programming proficiency and builds real-world cybersecurity awareness, making it a valuable learning experience in both secure coding and system design.

Table of Content

1. Introduction

1.1 Background of the selected problem

As digital platforms become a bigger part of our everyday lives, keeping our passwords safe is more important than ever. Unfortunately, many websites and apps still save passwords as plain text, which makes it easy for hackers to steal people's personal information. When passwords aren't protected properly, even a small security flaw can lead to serious problems.

By using simple encryption techniques even basic ones we can make stored passwords much harder to steal. Building a Password Encryption Tool in C is a practical way to show how even beginner-level cryptography and programming can make logins and personal data more secure. Not only does this help protect users, but it also teaches us why cybersecurity matters in software development today.

1.2 Industry Relevance

In today's professional world, secure password management isn't just a good practice—it's a necessity for keeping customer accounts, sensitive company data, and financial information safe. Businesses of all sizes are frequently targeted by cyber-attacks, especially when their password systems rely on weak or unencrypted storage. Even straightforward encryption methods can go a long way in protecting users and building trust.

Modern employers expect software developers to know how to write secure code and use safe authentication techniques in their products. By working on this project, we're tackling real problems industries face and learning the fundamentals of encryption and credential management—skills that are highly valued in the tech world. This kind of foundation prepares us for building reliable, secure applications that meet professional standards.

1.3 Aim & Objective of the Micro-Project

Aim

The goal of this project is to build a Password Encryption Tool in C that securely manages and stores user credentials. By combining simple encryption methods with essential C programming skills, the project aims to improve data security and lay the groundwork for safe authentication practices.

Objective:

1. Build a user-friendly C program that checks and encrypts passwords before saving them, so credentials are always kept safe.
2. Use file operations and smart searching to make storing and finding user accounts quick and dependable.
3. Show how everyday C programming skills—like using operators, control statements, functions, and string tricks—come together to make a practical, security-focused app that actually protects people's data.

1.4 Link to KITS Technology Mission

KITS has started a new initiative to make cyberspace safer and proactively detect cyber threats by implementing advanced technologies for building and defending systems. By using cybersecurity tools, automation, machine learning, and threat intelligence, they aim to protect digital environments more effectively. This mini-project makes a meaningful contribution to that mission by securely encrypting user data and credentials. Storing passwords safely helps lower the risk of security incidents and cyber alerts, directly supporting the goals of the college's technology mission.

2. Literature Review

Academic articles and journals stress that keeping passwords safe is vital for any software or system dealing with user data. Studies show that storing credentials in plain text can easily lead to security breaches and identity theft—making encryption or hashing a necessity.

Leading journals, such as the *International Journal of Information Security* and *ACM Computing Surveys*, note that while many large organizations use advanced cryptography, the basics are taught to students through simple techniques like substitution ciphers and XOR-based transformations. These straightforward methods help beginners understand how data protection and secure credential management work, even if they aren't used in high-security environments.

Research in programming education—including findings in *IEEE Transactions on Education*—highlights how problems in C programs often come from poor handling of strings, memory, or pointers. Recommended practices include always validating input, protecting buffers, and using modular code to prevent common bugs and security holes.

Technical articles also suggest adding file handling to student projects, which teaches how to safely store and manage data for long-term use. Real-life case studies have shown that combining basic encryption and modular programming techniques creates simple, effective systems for handling logins and credentials, providing meaningful practice in cybersecurity concepts.

Overall, the literature shows that building small tools like a Password Encryption Tool in C—complete with input checks, modular functions, file operations, and simple encryption—matches what is taught and recommended by experts. Projects like these help learners pick up crucial skills for creating safer software and managing credentials securely.

3. Methodology

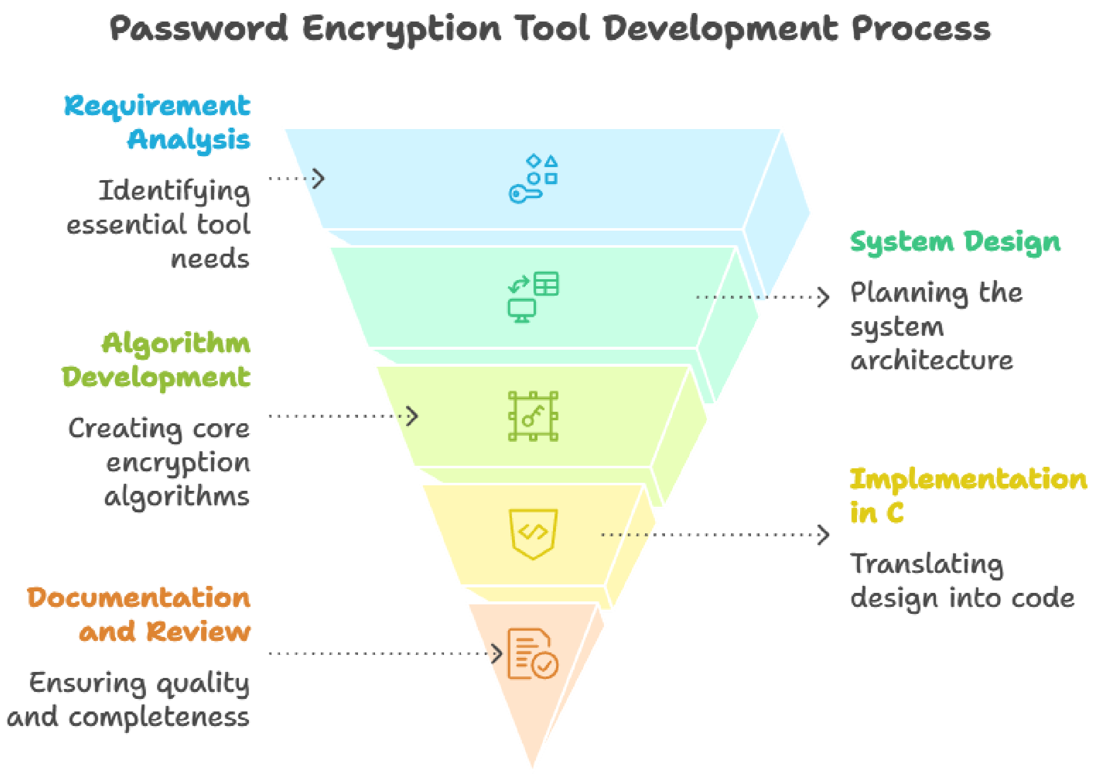

Password Encryption Tool Development Process
Requirement Analysis
Identifying essential tool needs
System Design
Planning the system architecture
Algorithm Development
Creating core encryption algorithms
Implementation in C
Translating design into code
Documentation and Review
Ensuring quality and completeness

Fig 3.1 Password Encryption Tool Development Process

1. Requirement Analysis

The process starts by looking closely at why password safety matters and what the project hopes to achieve. Since storing passwords as plain text can lead to security issues, adding encryption becomes essential. This stage is all about listing what the tool must do—like keeping credentials safe and easy to use—while also making sure it meets both practical needs and learning goals for the course.

1.1 Functional Requirements

- The tool enables users to create accounts by entering a username and password.
- Before encryption, passwords are checked for minimum length and other criteria to ensure validity.
- Passwords are encrypted using a basic Caesar cipher algorithm, as implemented in the code.

- Encrypted credentials are saved to a file, ensuring data can be persistently stored and retrieved.
- The system retrieves and decrypts stored credentials for user authentication, comparing values securely.
- A menu-driven interface provides user-friendly options to register new accounts, log in, view users, search credentials, sort entries, save/load data, and exit the program.

1.2 Non-Functional Requirements

1. Passwords must always be stored in encrypted form to ensure data safety and prevent exposure of plain text credentials.
2. The program should use clear and modular code—organizing logic into functions and structures for readability and easy maintenance.
3. Saved credential files should be persistent, allowing data to be accessed and managed over multiple sessions.
4. The system must operate efficiently, using minimal memory resources while maintaining fast performance for all user actions.

1.3 Mapping with Course Outcomes

- **CO1**: Operators, expressions, and control statements are applied for input validation, decision-making, and logic control.
- **CO2**: Searching techniques are used for locating usernames in stored files and verifying credentials.
- **CO3**: Functions and modular structure are used for separating encryption, validation, I/O, and verification processes.
- **CO4**: File handling and pointers are applied for persistent storage and secure access to encrypted credentials.
- **CO%**: Develop animation and gaming applications

4. SYSTEM DESIGN

4.1 Architectural Overview

The tool is organized into clear layers, each with a distinct role:

- **Input Layer:**
  Manages user interaction, guiding users to enter their usernames and passwords through the graphical interface (inputUsername, inputPassword). It ensures input fields are filled and formatted correctly.
- **Processing Layer:**
  Handles validation checks for passwords (such as minimum length) and applies the Caesar cipher encryption using the caesarEncrypt() function. This layer converts plain-text passwords into a secure, encoded format.
- **Storage Layer:**
  Oversees the saving and retrieval of user credentials. File operations in saveToFile() and loadFromFile() securely store and load encrypted user data, guaranteeing that records persist across sessions.
- **Authentication Layer:**
  Confirms user identity by searching for usernames with searchUserIndex(), decrypting stored passwords with caesarDecrypt(), and comparing them with login attempts in loginUser(). This provides reliable verification and prevents unauthorized access.

4.2 Module Breakdown

Each major task is represented as a separate function to follow structured programming:

1. Input Module
    - Uses C functions within the graphical interface to securely capture usernames and passwords (inputUsername, inputPassword).
    - Checks for empty or invalid inputs before allowing registration or login.
2. Validation Module
    - Confirms that passwords meet essential criteria, such as a minimum length (MIN_PASSWORD_LENGTH).

- Employs control statements to ensure that only valid inputs are processed.

3. Encryption Module
    - Uses the caesarEncrypt() function to protect passwords, applying a Caesar cipher that shifts each character.
    - Relies on loops and string manipulation to convert plain-text passwords into encrypted versions.
4. File Handling Module
    - Handles saving and loading data with saveToFile() and loadFromFile(), working with files in appropriate read/write modes.
    - Secures user credentials in persistent storage and retrieves them for authentication.
5. Authentication Module
    - Encrypts login attempts for verification using built-in functions (loginUser(), caesarDecrypt()).
    - Searches existing user records and matches credentials to ensure secure access, displaying clear feedback for success or errors.

4.3 Data Flow Design

The flow of data moves through stages:

The movement of data in the tool follows these key stages:

1. **User Input → Input Module:**
   Users enter their username and password through the graphical interface, and these details are collected by the input module in your code.
2. **Input → Validation → Encryption:**
   The entered password is checked to ensure it meets the minimum requirements. Once validated, it is encrypted using the Caesar cipher algorithm
   (caesarEncrypt() function).
3. **Encrypted Data → File Storage:**
   The encrypted password, along with the username, is saved to a persistent file using the saveToFile() function, ensuring credentials are stored securely for future access.
4. **Login Attempt → Encryption → Search → Match/Fail:**
   During login, the tool takes the inputted username and password, re-encrypts the password, and searches for a matching username in the stored records

(searchUserIndex()). If the encrypted input matches the stored credential, access is granted; otherwise, the login fails and an error message is shown.

5. Algorithm Development

5.1 Password Validation Algorithm

- Accepts the password from the inputPassword string.
- Checks that the password length meets the minimum requirement (MIN_PASSWORD_LENGTH, set to 6).
- Flags the input as invalid if the password is too short or the field is empty.
- Returns feedback through a message and does not proceed with registration until a valid password is entered.

5.2 Encryption Algorithm

A simple encryption logic was selected due to course limitations:

- Iterates through each character of the entered password in the caesarEncrypt() function.
- Applies a Caesar cipher by shifting letters (using ENCRYPTION_KEY) and digits, while leaving other symbols unchanged.
- Builds the encrypted password string, which is then stored instead of the plain text.
- This process ensures that passwords saved in files are not directly readable, adding a layer of security to the stored data.

5.3 Storage Algorithm

- Opens the storage file (users.dat) in write mode using the saveToFile() function.
- Writes all registered users' usernames and their corresponding encrypted passwords, along with an active status flag, to the file.
- Closes the file securely after saving to ensure data integrity and prevent loss or corruption.
- The storage method keeps credentials persistent, allowing reliable user management across sessions.

5.4 Credential Verification Algorithm

- **Opening the file:** The program loads user credentials from "users.dat" using the loadFromFile() function when starting, or after requesting to load data.

- **Searching usernames:** The searchUserIndex() function examines each record in the loaded user array to find a matching username.
- **Encrypting input password:** During login, the entered password is encrypted on the fly using caesarEncrypt() or decrypted using caesarDecrypt() for matching against stored data.
- **Comparing encrypted data:** The encrypted version of the input password is compared to the stored encrypted password for that user within loginUser().
- **Returning authentication status:** The outcome (success or failure) is shown to the user with feedback messages, as managed by showMessage().

5.5 Error Handling Strategy

- **Incorrect File Paths:**
  When the storage file (users.dat) can't be opened for reading or writing, error messages are displayed using the showMessage() function (for example, "Error opening file!" or "No saved data found!").

- **Missing Data:**
  If the file is empty or if there are no registered users, informative feedback is provided ("No users registered yet!" or "No saved data found!"), helping the user understand the issue.

- **Invalid Input:**
  The program checks if usernames and passwords are left blank or if the password is too short. It warns users with clear messages such as "Please fill all fields!" or "Password too short!" before proceeding.

- **Unexpected Termination:**
  The use of menu controls and window-close checks
  with CloseWindow() and WindowShouldClose() helps prevent abrupt exits and handles application termination gracefully, ensuring user data is saved and feedback is given.

6. Implementation

6.1 Use of C Fundamentals

Implementation relies heavily on:

- **Control Statements:**
  if and *switch* statements are used throughout, handling menu navigation, input validation, and responses to user actions.
- **Loops:**
  for loops iterate over each password character for encryption/decryption, and traverse arrays to search or sort users.
- **Operators:**
  Arithmetic and logical operators facilitate password transformation in the Caesar cipher and comparisons during searches and validation.
- **Strings:**
  Character arrays and string functions (like strcpy, strcmp, strlen) are central for managing usernames and passwords efficiently.
- **Functions:**
  All major tasks are separated into distinct functions (registerUser, loginUser, caesarEncrypt, saveToFile, etc.), making the program easy to read, test, and maintain.

6.2 Use of Pointers

Pointers are required for:

1. **File Pointer Operations:**
   Pointers to FILE objects (e.g., FILE *fp;) are used for opening, reading from, and writing to data files in functions like saveToFile() and loadFromFile().
2. **Iterating Through Characters:**
   Character pointers and array indexing are employed in the encryption and decryption routines (caesarEncrypt, caesarDecrypt), efficiently traversing each password character.

3. **Passing Strings to Functions:**
   String arguments are passed by pointer to various functions—such as password and username arrays sent to caesarEncrypt(), registerUser(), and others—allowing direct manipulation and transformation of user data.

6.3 File Handling Implementation

- Files are managed using standard C file handling functions:
    - fopen() is used to open the storage file (users.dat) for reading or writing in saveToFile() and loadFromFile().
    - fprintf() and fscanf() are applied for writing user credentials to the file and reading them back, ensuring data is stored and retrieved in the correct format.
    - fclose() closes the file after operations are complete, protecting against data loss and corruption.

7. Testing & Evaluation

The following tests have been done:

1. Function Testing
2. Input Testing
3. File Testing
4. Security Testing
5. Performance Testing

Resources Required

S. No.	Resource	Suggested Broad Specification	Quantity
1	Laptop	Minimum Intel 5 Processor	1 No.
2	VS Code	1. GCC (Compiler) 2. raylib (GUI Library) 3.	Open Source
3	Github	3.17	Open Source

8. Source Code

Project Structure:

```
PassGuard/
├── .vscode/
│   └── launch.json
├── githubAssets/
│   ├── banner.jpg
│   ├── errorMessage.gif
│   ├── loadsaveUsers.gif
│   ├── Main Menu.png
│   ├── searchUsers.gif
│   ├── userLogin.gif
│   ├── userRegistration.gif
│   └── viewUsers.png
├── Project Documentation/
│   └── URK25CS1106 - (CodeGen PPS Micro Project PPT).pdf
├── passguard.c
├── passguard.exe
├── README.md
└── users.dat
```

Github Repository Link: https://github.com/RM1338/PassGuard

Main Program:

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include "raylib.h"


// Pre-processor directives for configurable security features

#define MAX_PASSWORD_LENGTH 50

#define ENCRYPTION_KEY 5

#define MAX_USERS 10

#define MIN_PASSWORD_LENGTH 6

#define SCREEN_WIDTH 1000

#define SCREEN_HEIGHT 700


// Structure to store user credentials (CO3 - Structured Programming)

typedef struct {

    char username[50];

    char encrypted_password[MAX_PASSWORD_LENGTH];

    int is_active;

} User;


// Global array to store users (CO4 - Memory Management)

User users[MAX_USERS];
```

```
int user_count = 0;


// UI State variables
int currentScreen = 0; // 0=Menu, 1=Register, 2=Login, 3=View, 4=Search
char inputUsername[50] = {0};
char inputPassword[MAX_PASSWORD_LENGTH] = {0};
char searchUsername[50] = {0};
char messageText[200] = {0};
int messageType = 0; // 0=none, 1=success, 2=error
float messageTimer = 0;
int activeInput = 0; // 0=none, 1=username, 2=password


// Function prototypes
void caesarEncrypt(char *password, char *encrypted, int key);
void caesarDecrypt(char *encrypted, char *decrypted, int key);
int hashPassword(char *password);
void registerUser();
void loginUser();
void searchUser();
void bubbleSortUsers();
int searchUserIndex(char *username);
void saveToFile();
void loadFromFile();
void drawMainMenu();
void drawRegisterScreen();
void drawLoginScreen();

void drawViewScreen();

void drawSearchScreen();

void showMessage(const char* text, int type);

void drawButton(Rectangle bounds, const char* text, Color color);

int isButtonPressed(Rectangle bounds);


// Caesar Cipher Encryption (CO1 - Expressions and Operators)

void caesarEncrypt(char *password, char *encrypted, int key) {

    int i;

    for (i = 0; password[i] != '\0'; i++) {

        if (isalpha(password[i])) {

            if (isupper(password[i])) {

                encrypted[i] = ((password[i] - 'A' + key) % 26) + 'A';

            } else {

                encrypted[i] = ((password[i] - 'a' + key) % 26) + 'a';

            }

        } else if (isdigit(password[i])) {

            encrypted[i] = ((password[i] - '0' + key) % 10) + '0';

        } else {

            encrypted[i] = password[i];

        }

    }

    encrypted[i] = '\0';

}
```

```
// Caesar Cipher Decryption
void caesarDecrypt(char *encrypted, char *decrypted, int key) {
    int i;
    for (i = 0; encrypted[i] != '\0'; i++) {
        if (isalpha(encrypted[i])) {
            if (isupper(encrypted[i])) {
                decrypted[i] = ((encrypted[i] - 'A' - key + 26) % 26) + 'A';
            } else {
                decrypted[i] = ((encrypted[i] - 'a' - key + 26) % 26) + 'a';
            }
        } else if (isdigit(encrypted[i])) {
            decrypted[i] = ((encrypted[i] - '0' - key + 10) % 10) + '0';
        } else {
            decrypted[i] = encrypted[i];
        }
    }
    decrypted[i] = '\0';
}

// Simple Hash Function (CO1 - Expressions)
int hashPassword(char *password) {
    int hash = 0;
    int i;
    for (i = 0; password[i] != '\0'; i++) {
        hash += password[i] * (i + 1);
    }
    return hash % 1000;
}

// Linear Search for user (CO2 - Searching Techniques)
int searchUserIndex(char *username) {
    int i;
    for (i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0 && users[i].is_active) {
            return i;
        }
    }
    return -1;
}

// Register new user (CO3 - Structured Programming)
void registerUser() {
    if (user_count >= MAX_USERS) {
        showMessage("User limit reached!", 2);
        return;
    }

    if (strlen(inputUsername) == 0 || strlen(inputPassword) == 0) {
        showMessage("Please fill all fields!", 2);
        return;
    }

    // Check if user already exists (CO2 - Searching)
    if (searchUserIndex(inputUsername) != -1) {
        showMessage("Username already exists!", 2);
        return;
    }

    // Validate password length (CO1 - Control Statements)
    if (strlen(inputPassword) < MIN_PASSWORD_LENGTH) {
        showMessage("Password too short! Minimum 6 characters.", 2);
        return;
    }

    // Store user data (CO3 - Structures)
    strcpy(users[user_count].username, inputUsername);
    caesarEncrypt(inputPassword, users[user_count].encrypted_password,
ENCRYPTION_KEY);
    users[user_count].is_active = 1;

    showMessage("User registered successfully!", 1);

    // Clear inputs
    memset(inputUsername, 0, sizeof(inputUsername));
    memset(inputPassword, 0, sizeof(inputPassword));

    user_count++;
}

// Login user (Verify credentials)
void loginUser() {
    char decrypted[MAX_PASSWORD_LENGTH];
    int index;

    if (strlen(inputUsername) == 0 || strlen(inputPassword) == 0) {
        showMessage("Please fill all fields!", 2);
        return;
    }

    // Search for user (CO2 - Searching)
    index = searchUserIndex(inputUsername);

    if (index == -1) {
        showMessage("User not found!", 2);
        return;
    }

    // Decrypt stored password and verify (CO1 - Control Statements)
    caesarDecrypt(users[index].encrypted_password, decrypted, ENCRYPTION_KEY);
    if (strcmp(inputPassword, decrypted) == 0) {
        char msg[100];
        sprintf(msg, "Login Successful! Welcome, %s!", inputUsername);
        showMessage(msg, 1);

        // Clear inputs
        memset(inputUsername, 0, sizeof(inputUsername));
        memset(inputPassword, 0, sizeof(inputPassword));
    } else {
        showMessage("Incorrect password!", 2);
    }
}

// Search user
void searchUser() {
    int index = searchUserIndex(searchUsername);
    if (index != -1) {
        char msg[100];
        sprintf(msg, "User found: %s", users[index].username);
        showMessage(msg, 1);
    } else {
        showMessage("User not found!", 2);
    }
}
```

```
// Bubble Sort users by username (CO2 - Sorting Techniques)
void bubbleSortUsers() {
    int i, j;
    User temp;

    for (i = 0; i < user_count - 1; i++) {
        for (j = 0; j < user_count - i - 1; j++) {
            if (strcmp(users[j].username, users[j + 1].username) > 0) {
                // Swap users
                temp = users[j];
                users[j] = users[j + 1];
                users[j + 1] = temp;
            }
        }
    }

    showMessage("Users sorted alphabetically!", 1);
}

// Save data to file (CO4 - File Handling)
void saveToFile() {
    FILE *fp;
    int i;
    fp = fopen("users.dat", "w");


    if (fp == NULL) {
        showMessage("Error opening file!", 2);
        return;
    }
    fprintf(fp, "%d\n", user_count);
    for (i = 0; i < user_count; i++) {
        fprintf(fp, "%s %s %d\n",
                users[i].username,
                users[i].encrypted_password,
                users[i].is_active);
    }
    fclose(fp);
    showMessage("Data saved to file successfully!", 1);
}


// Load data from file (CO4 - File Handling)
void loadFromFile() {
    FILE *fp;
    int i;
    fp = fopen("users.dat", "r");
    if (fp == NULL) {
        showMessage("No saved data found!", 2);
        return;
    }
    fscanf(fp, "%d", &user_count);
    for (i = 0; i < user_count; i++) {
        fscanf(fp, "%s %s %d",
               users[i].username,
               users[i].encrypted_password,
               &users[i].is_active);
    }
    fclose(fp);
    showMessage("Data loaded successfully!", 1);
}

// Show message
void showMessage(const char* text, int type) {
    strcpy(messageText, text);
    messageType = type;
    messageTimer = 3.0f; // 3 seconds
}

// Draw button
void drawButton(Rectangle bounds, const char* text, Color color) {
    DrawRectangleRounded(bounds, 0.3f, 10, color);

    int textWidth = MeasureText(text, 20);
    DrawText(text, bounds.x + (bounds.width - textWidth) / 2,
            bounds.y + (bounds.height - 20) / 2, 20, WHITE);
}

// Check if button is pressed
int isButtonPressed(Rectangle bounds) {
    Vector2 mousePos = GetMousePosition();
    if (CheckCollisionPointRec(mousePos, bounds)) {
        DrawRectangleRounded(bounds, 0.3f, 10, ColorAlpha(WHITE, 0.2f));
        if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
            return 1;
        }
    }
    return 0;
}

// Draw Main Menu
void drawMainMenu() {
    DrawText("PASSWORD ENCRYPTION & CREDENTIAL MANAGER", 150, 50, 30,
DARKBLUE);
    DrawText("Cybersecurity Micro Project", 350, 90, 20, GRAY);

    Rectangle btn1 = {150, 150, 250, 60};
    Rectangle btn2 = {450, 150, 250, 60};
    Rectangle btn3 = {750, 150, 200, 60};
    Rectangle btn4 = {150, 240, 250, 60};
    Rectangle btn5 = {450, 240, 250, 60};
    Rectangle btn6 = {750, 240, 200, 60};
    Rectangle btn7 = {150, 330, 250, 60};
    Rectangle btn8 = {450, 330, 250, 60};

    drawButton(btn1, "Register User", DARKGREEN);
    drawButton(btn2, "Login", DARKBLUE);
    drawButton(btn3, "View Users", DARKPURPLE);
    drawButton(btn4, "Search User", ORANGE);
    drawButton(btn5, "Sort Users", MAROON);
    drawButton(btn6, "Save Data", DARKGRAY);
    drawButton(btn7, "Load Data", BROWN);
    drawButton(btn8, "Exit", RED);

    if (isButtonPressed(btn1)) currentScreen = 1;
    if (isButtonPressed(btn2)) currentScreen = 2;
    if (isButtonPressed(btn3)) currentScreen = 3;
    if (isButtonPressed(btn4)) currentScreen = 4;
    if (isButtonPressed(btn5)) { bubbleSortUsers(); currentScreen = 3; }
    if (isButtonPressed(btn6)) saveToFile();
    if (isButtonPressed(btn7)) loadFromFile();
    if (isButtonPressed(btn8)) CloseWindow();

    // Stats
    DrawText(TextFormat("Total Users: %d", user_count), 150, 450, 25, DARKBLUE);
```

```
    DrawText(TextFormat("Encryption Key: %d", ENCRYPTION_KEY), 150, 490, 25,
DARKBLUE);
}

// Draw Register Screen
void drawRegisterScreen() {
    DrawText("REGISTER NEW USER", 350, 50, 30, DARKGREEN);

    DrawText("Username:", 200, 150, 25, BLACK);
    Rectangle usernameBox = {200, 190, 600, 40};
    DrawRectangleRec(usernameBox, activeInput == 1 ? LIGHTGRAY : WHITE);
    DrawRectangleLinesEx(usernameBox, 2, activeInput == 1 ? DARKGREEN : GRAY);
    DrawText(inputUsername, 210, 200, 20, BLACK);

    DrawText("Password:", 200, 260, 25, BLACK);
    Rectangle passwordBox = {200, 300, 600, 40};
    DrawRectangleRec(passwordBox, activeInput == 2 ? LIGHTGRAY : WHITE);
    DrawRectangleLinesEx(passwordBox, 2, activeInput == 2 ? DARKGREEN : GRAY);

    // Draw masked password
    char masked[MAX_PASSWORD_LENGTH];
    int i;
    for (i = 0; i < strlen(inputPassword); i++) {
        masked[i] = '*';
    }
    masked[i] = '\0';
    DrawText(masked, 210, 310, 20, BLACK);

    // Handle input
    if (isButtonPressed(usernameBox)) activeInput = 1;
    if (isButtonPressed(passwordBox)) activeInput = 2;

    int key = GetCharPressed();
    while (key > 0) {
        if (activeInput == 1 && strlen(inputUsername) < 49) {
            int len = strlen(inputUsername);
            inputUsername[len] = (char)key;
            inputUsername[len + 1] = '\0';
        } else if (activeInput == 2 && strlen(inputPassword) < MAX_PASSWORD_LENGTH
- 1) {
            int len = strlen(inputPassword);
            inputPassword[len] = (char)key;
            inputPassword[len + 1] = '\0';
        }
        key = GetCharPressed();
    }

    if (IsKeyPressed(KEY_BACKSPACE)) {
        if (activeInput == 1) {
            int len = strlen(inputUsername);
            if (len > 0) inputUsername[len - 1] = '\0';
        } else if (activeInput == 2) {
            int len = strlen(inputPassword);
            if (len > 0) inputPassword[len - 1] = '\0';
        }
    }

    Rectangle registerBtn = {300, 400, 150, 50};
    Rectangle backBtn = {500, 400, 150, 50};

    drawButton(registerBtn, "Register", DARKGREEN);
    drawButton(backBtn, "Back", GRAY);

    if (isButtonPressed(registerBtn)) registerUser();
    if (isButtonPressed(backBtn)) {
        currentScreen = 0;
        memset(inputUsername, 0, sizeof(inputUsername));
        memset(inputPassword, 0, sizeof(inputPassword));
        activeInput = 0;
    }
}

// Draw Login Screen
void drawLoginScreen() {
    DrawText("USER LOGIN", 400, 50, 30, DARKBLUE);
    DrawText("Username:", 200, 150, 25, BLACK);
    Rectangle usernameBox = {200, 190, 600, 40};
    DrawRectangleRec(usernameBox, activeInput == 1 ? LIGHTGRAY : WHITE);
    DrawRectangleLinesEx(usernameBox, 2, activeInput == 1 ? DARKBLUE : GRAY);
    DrawText(inputUsername, 210, 200, 20, BLACK);

    DrawText("Password:", 200, 260, 25, BLACK);
    Rectangle passwordBox = {200, 300, 600, 40};
    DrawRectangleRec(passwordBox, activeInput == 2 ? LIGHTGRAY : WHITE);
    DrawRectangleLinesEx(passwordBox, 2, activeInput == 2 ? DARKBLUE : GRAY);

    char masked[MAX_PASSWORD_LENGTH];
    int i;
    for (i = 0; i < strlen(inputPassword); i++) {
        masked[i] = '*';
    }
    masked[i] = '\0';
    DrawText(masked, 210, 310, 20, BLACK);

    if (isButtonPressed(usernameBox)) activeInput = 1;
    if (isButtonPressed(passwordBox)) activeInput = 2;

    int key = GetCharPressed();
    while (key > 0) {
        if (activeInput == 1 && strlen(inputUsername) < 49) {
            int len = strlen(inputUsername);
            inputUsername[len] = (char)key;
            inputUsername[len + 1] = '\0';
        } else if (activeInput == 2 && strlen(inputPassword) < MAX_PASSWORD_LENGTH
- 1) {
            int len = strlen(inputPassword);
            inputPassword[len] = (char)key;
            inputPassword[len + 1] = '\0';
        }
        key = GetCharPressed();
    }

    if (IsKeyPressed(KEY_BACKSPACE)) {
        if (activeInput == 1) {
            int len = strlen(inputUsername);
            if (len > 0) inputUsername[len - 1] = '\0';
        } else if (activeInput == 2) {
            int len = strlen(inputPassword);
            if (len > 0) inputPassword[len - 1] = '\0';
        }
    }

    Rectangle loginBtn = {300, 400, 150, 50};
    Rectangle backBtn = {500, 400, 150, 50};
    drawButton(loginBtn, "Login", DARKBLUE);
    drawButton(backBtn, "Back", GRAY);

    if (isButtonPressed(loginBtn)) loginUser();
    if (isButtonPressed(backBtn)) {
        currentScreen = 0;
        memset(inputUsername, 0, sizeof(inputUsername));
        memset(inputPassword, 0, sizeof(inputPassword));
        activeInput = 0;
    }
}

// Draw View Screen
void drawViewScreen() {
    DrawText("ALL REGISTERED USERS", 350, 30, 30, DARKPURPLE);

    if (user_count == 0) {
        DrawText("No users registered yet!", 350, 300, 25, GRAY);
    } else {
        DrawText("No.", 50, 100, 20, BLACK);
        DrawText("Username", 150, 100, 20, BLACK);
        DrawText("Encrypted Password", 350, 100, 20, BLACK);
        DrawText("Hash", 650, 100, 20, BLACK);
        DrawLine(50, 130, 950, 130, BLACK);


        int i;
        char decrypted[MAX_PASSWORD_LENGTH];
        for (i = 0; i < user_count && i < 10; i++) {
            if (users[i].is_active) {
                caesarDecrypt(users[i].encrypted_password, decrypted, ENCRYPTION_KEY);
                DrawText(TextFormat("%d", i + 1), 50, 150 + i * 40, 20, BLACK);
                DrawText(users[i].username, 150, 150 + i * 40, 20, BLACK);
                DrawText(users[i].encrypted_password, 350, 150 + i * 40, 20, BLACK);
                DrawText(TextFormat("%d", hashPassword(decrypted)), 650, 150 + i * 40, 20,
BLACK);
            }
        }
    }


    Rectangle backBtn = {425, 600, 150, 50};
    drawButton(backBtn, "Back", GRAY);


    if (isButtonPressed(backBtn)) currentScreen = 0;
}


// Draw Search Screen
void drawSearchScreen() {
    DrawText("SEARCH USER", 400, 50, 30, ORANGE);
DrawText("Enter Username:", 200, 150, 25, BLACK);
Rectangle searchBox = {200, 190, 600, 40};
DrawRectangleRec(searchBox, activeInput == 1 ? LIGHTGRAY : WHITE);
DrawRectangleLinesEx(searchBox, 2, activeInput == 1 ? ORANGE : GRAY);
DrawText(searchUsername, 210, 200, 20, BLACK);

if (isButtonPressed(searchBox)) activeInput = 1;

int key = GetCharPressed();
while (key > 0) {
   if (activeInput == 1 && strlen(searchUsername) < 49) {
      int len = strlen(searchUsername);
      searchUsername[len] = (char)key;
      searchUsername[len + 1] = '\0';
   }
   key = GetCharPressed();
}

if (IsKeyPressed(KEY_BACKSPACE) && activeInput == 1) {
   int len = strlen(searchUsername);
   if (len > 0) searchUsername[len - 1] = '\0';
}

Rectangle searchBtn = {300, 300, 150, 50};
```

```
    Rectangle backBtn = {500, 300, 150, 50};


    drawButton(searchBtn, "Search", ORANGE);
    drawButton(backBtn, "Back", GRAY);


    if (isButtonPressed(searchBtn)) searchUser();
    if (isButtonPressed(backBtn)) {
        currentScreen = 0;
        memset(searchUsername, 0, sizeof(searchUsername));
        activeInput = 0;
    }
}
// Main function
int main() {
    InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Password Encryption Tool -
Cybersecurity Project");
    SetTargetFPS(60);


    // Main loop
    while (!WindowShouldClose()) {
        // Update message timer
        if (messageTimer > 0) {
            messageTimer -= GetFrameTime();
        }
        BeginDrawing();
        ClearBackground(RAYWHITE);

        // Draw current screen (CO1 - Control Statements)

        switch (currentScreen) {

            case 0: drawMainMenu(); break;

            case 1: drawRegisterScreen(); break;

            case 2: drawLoginScreen(); break;

            case 3: drawViewScreen(); break;

            case 4: drawSearchScreen(); break;

        }

        // Draw message

        if (messageTimer > 0) {

            Color msgColor = (messageType == 1) ? DARKGREEN : RED;

            Color bgColor = (messageType == 1) ? ColorAlpha(GREEN, 0.3f) :
ColorAlpha(RED, 0.3f);


            Rectangle msgBox = {200, SCREEN_HEIGHT - 80, 600, 50};

            DrawRectangleRec(msgBox, bgColor);

            DrawRectangleLinesEx(msgBox, 2, msgColor);


            int textWidth = MeasureText(messageText, 20);

            DrawText(messageText, 200 + (600 - textWidth) / 2, SCREEN_HEIGHT - 65, 20,
msgColor);

        }

        EndDrawing();

    }
    CloseWindow();

    return 0;

}
```

9. Results & Interpretation

Results:

The Password Encryption Tool was created in C and features a simple menu-driven interface where users can easily register and log in. Whenever a user sets a password, it’s instantly scrambled using a Caesar cipher so what gets saved isn’t readable as plain text. Later, when logging in, the tool encrypts the new password entry the same way and checks it against what’s stored. This setup makes sure only the right credentials get through, and keeps sensitive data safe behind the scenes.

Interpretation:

The outcome of the project clearly illustrates that even simple encryption like the Caesar cipher used here adds a valuable layer of security to password management in C. By converting passwords into unreadable strings, the tool avoids storing them in plain text, reducing immediate risks from exposure. Every login uses encrypted comparison instead of checking raw inputs, keeping the verification process confidential.

Your code demonstrates how cryptographic concepts can be introduced through practical programming elements functions, loops, and string operations all organized within a modular structure. Each task, from user input to encryption and authentication, is handled by separate functions, making the program easier to maintain and extend.

While this form of encryption isn’t suited for advanced security needs, it gives a hands-on introduction to protecting credentials and structuring authentication systems. The mini-project fulfils key learning objectives in structured programming, modular function design, string manipulation, and basic security logic using C, forming a strong foundation for further exploration in cybersecurity.

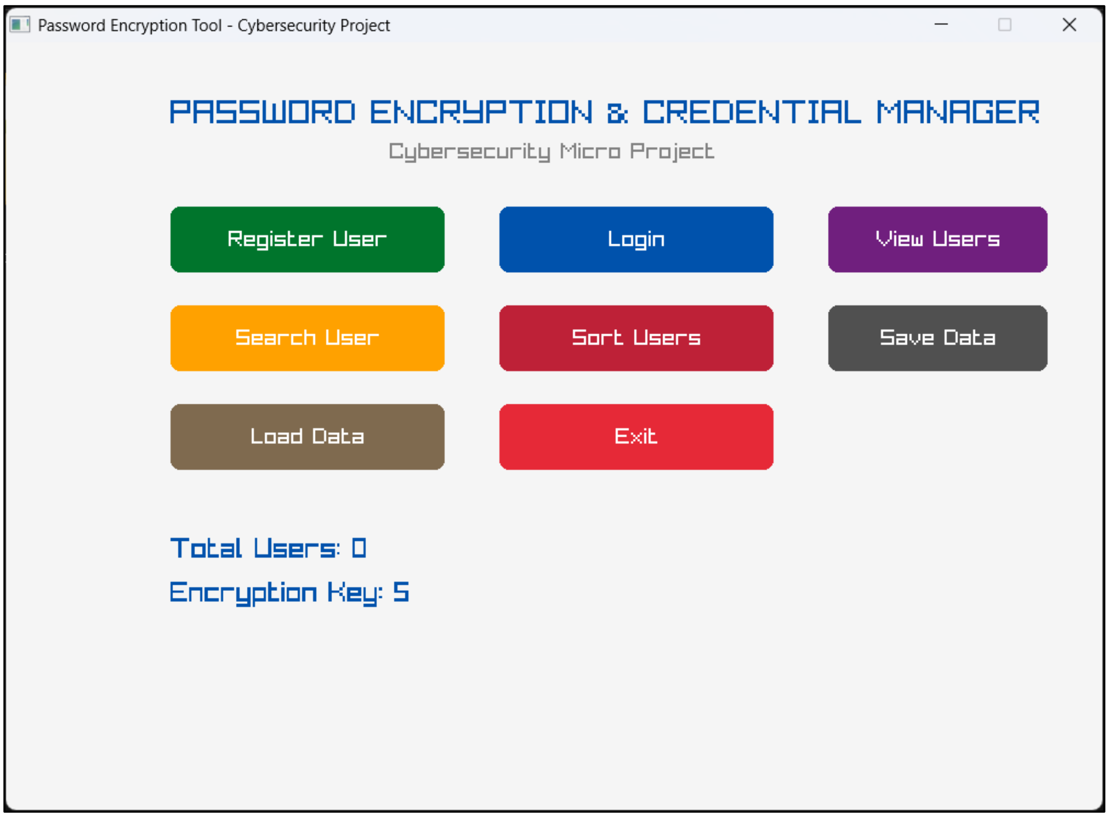

Password Encryption Tool - Cybersecurity Project
PASSWORD ENCRYPTION & CREDENTIAL MANAGER
Cybersecurity Micro Project
Register User
Login
View Users
Search User
Sort Users
Save Data
Load Data
Exit
Total Users: 0
Encryption Key: 5

Fig 9.1 Main Menu

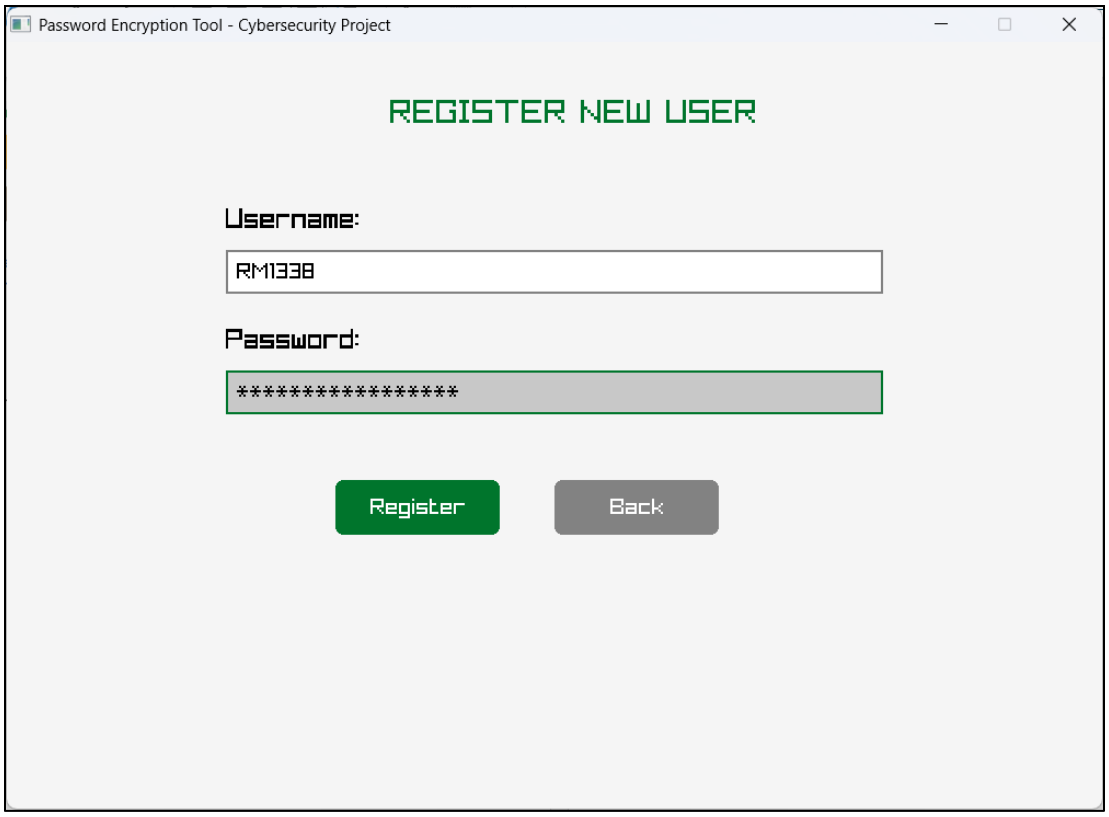


Fig 9.2 User Registration

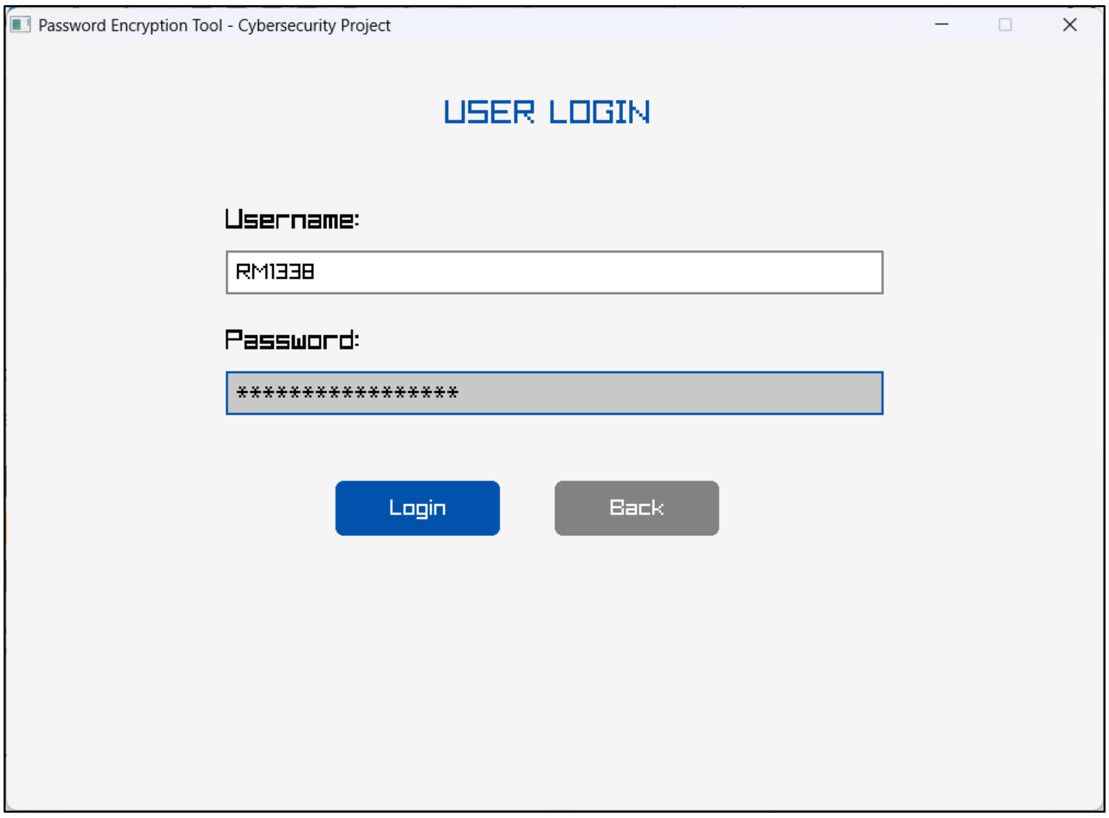
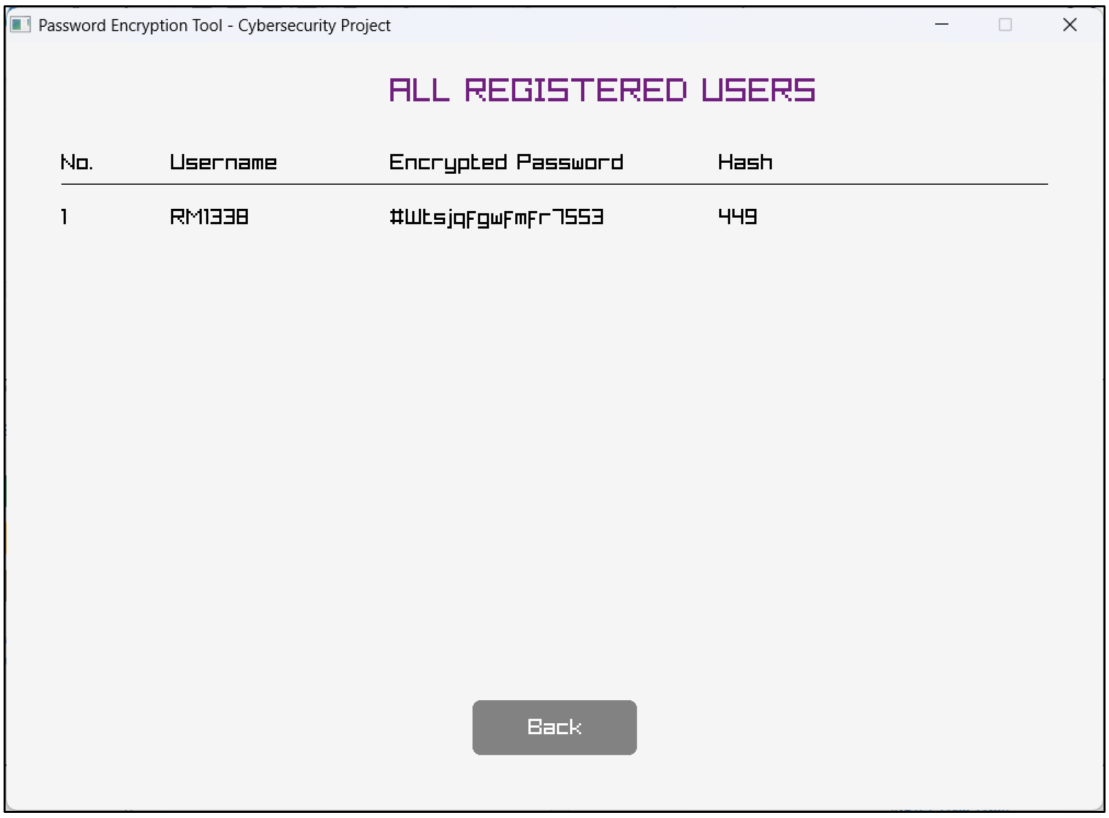
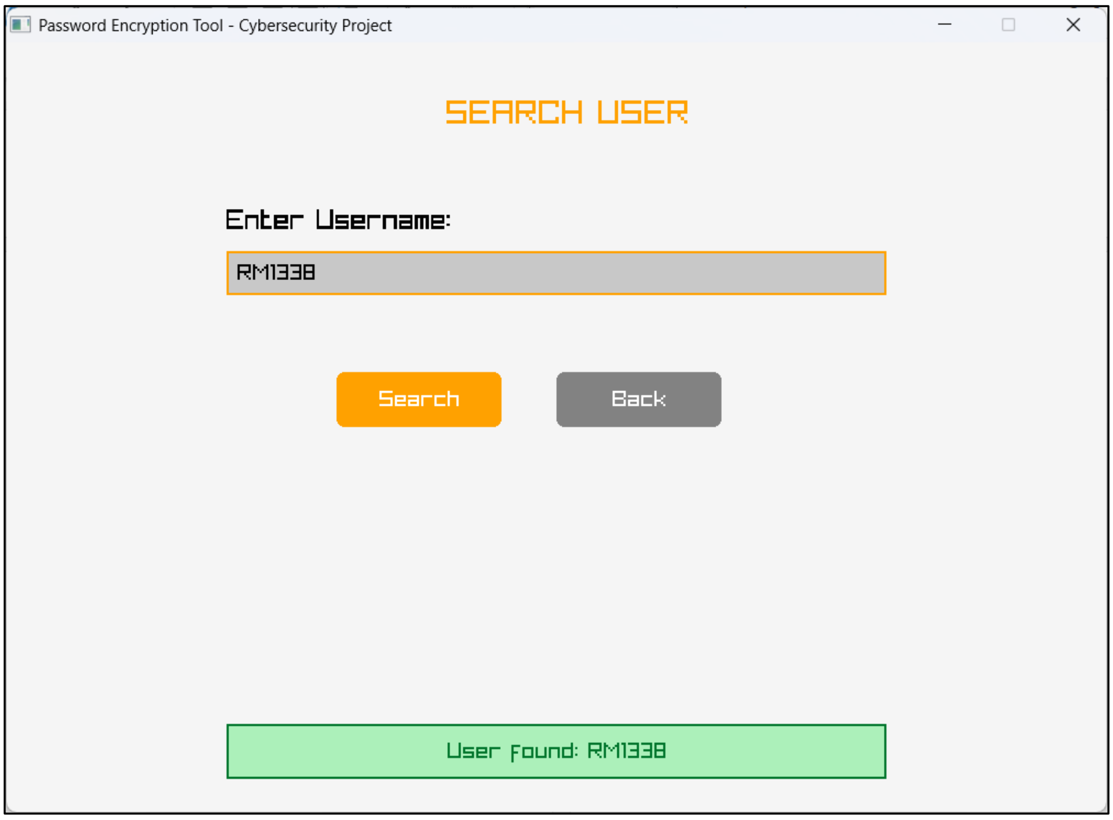


Fig 9.3 User Login



No.	Username	Encrypted Password	Hash
1	RM1338	#WtsjqFgwFmFr7553	449

Fig 9.4 View Users


Password Encryption Tool - Cybersecurity Project

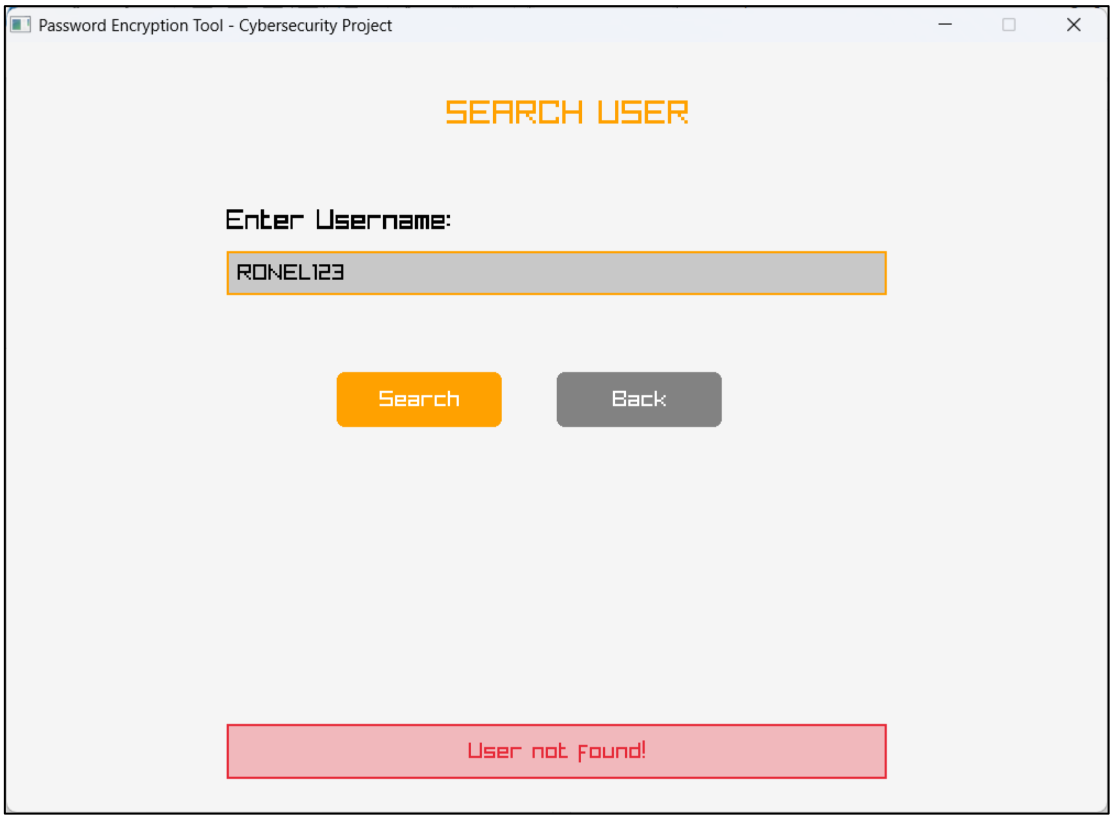
SEARCH USER

Enter Username:

RM1338

User found: RM1338

Fig 9.5 Search Users (User Found)


Password Encryption Tool - Cybersecurity Project

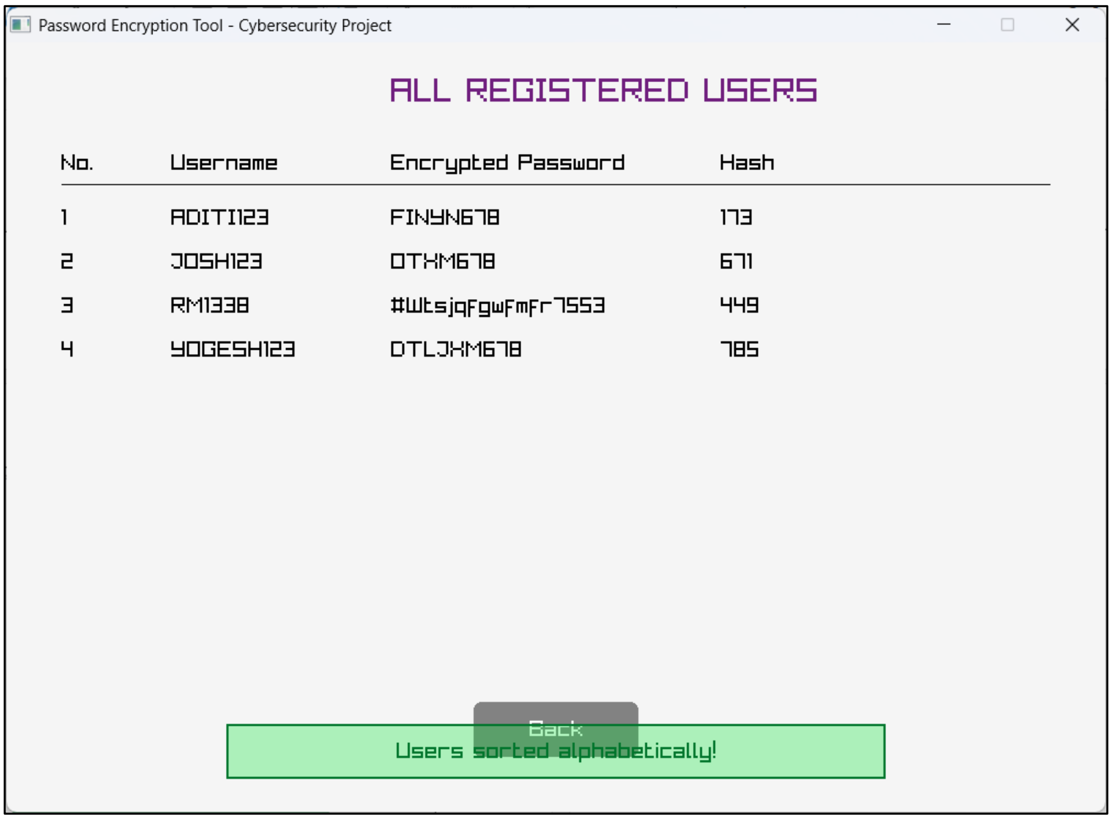
SEARCH USER

Enter Username:

RONEL123

User not found!

Fig 9.5 Search Users (User Found)

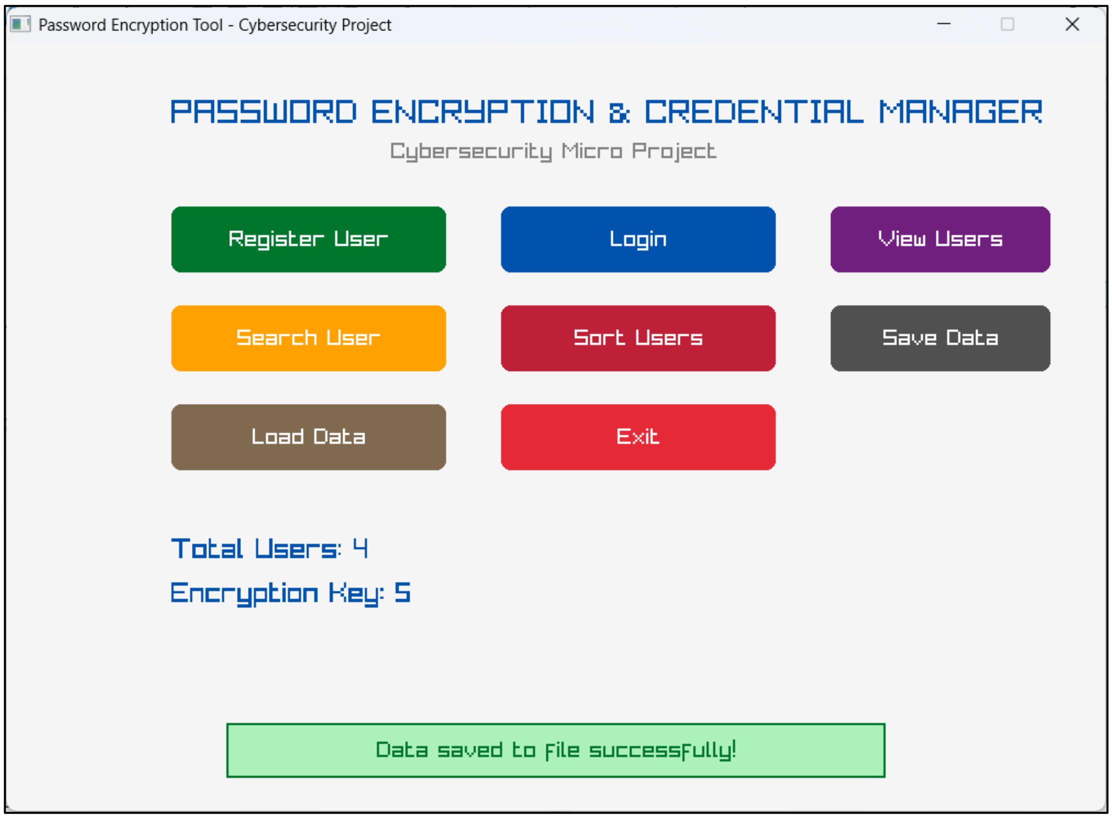

Password Encryption Tool - Cybersecurity Project

ALL REGISTERED USERS

No.	Username	Encrypted Password	Hash
1	ADITI123	FINYN678	173
2	JOSH123	OTXM678	671
3	RM1338	#WtsjqFgwFmFr7553	449
4	YOGESH123	DTLJXM678	785

Users sorted alphabetically!

Fig 9.6 Sort Users (Alphabetically)


Password Encryption Tool - Cybersecurity Project

PASSWORD ENCRYPTION & CREDENTIAL MANAGER

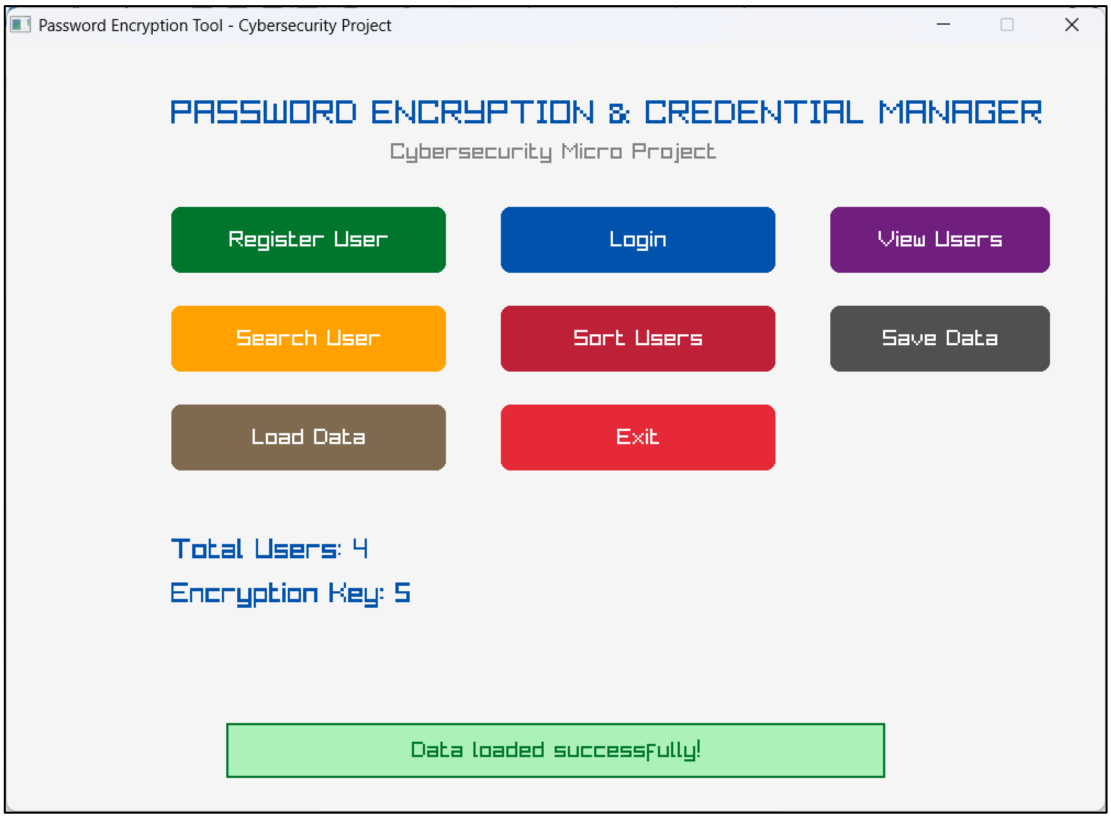
Cybersecurity Micro Project

Total Users: 4

Encryption Key: 5

Data saved to file successfully!

Fig 9.7 Save Date (Data saved to users.dat)


Password Encryption Tool - Cybersecurity Project

PASSWORD ENCRYPTION & CREDENTIAL MANAGER

Cybersecurity Micro Project

Total Users: 4

Encryption Key: 5

Data loaded successfully!

Fig 9.8 Date Loaded (Data loaded from users.dat)

10. Conclusion & Future Scope

Conclusion:

This project's Password Encryption Tool demonstrates basic concepts of the C programming language, including operators, control statements, loops, string handling, file management, structures, and modular programming. The project enhances user interaction by implementing a basic encryption technique and integrating simple graphics using the raylib library, but it also shows how one understands secure coding practices.

On the whole, it reaches its principal objective: to provide a user-friendly credential manager for beginners that securely stores user details and allows students to reflect on the importance of security while programming.

This project helped strengthen problem-solving skills, logical thinking, and the practical application of several Course Outcomes (COs). It has aligned very well with the learning goals of first-year CSE students.

Future Scope:

Stronger encryption algorithms

The current system uses a simple encryption method that is suitable for learning. In the future, this can be upgraded to more secure algorithms, such as AES, RSA, or hashing techniques like SHA-256.

Improvement to the Graphical User Interface

Advanced yet easy-to-follow graphics may be added, such as buttons, animations, or mouse interactions, that give the user ease.

Password Strength Checker

This tool can also be extended to check password strength on the basis of length, character variety, and common patterns to help users generate much better passwords.

Multi-user Database System This system can be scaled up with more users by integrating it into a proper database, such as SQLite or JSON-based storage. Two-Factor Authentication Performing OTP-based verification or verification via email can enhance the security level of the login process. Cross-Platform Compatibility It can be further worked on to run seamlessly on Windows, Linux, and macOS, with uniform performance regarding graphics and file operations. Real-time Error Handling & Logging In the future, detailed logs or error messages could be added to identify login failures, mistakes in operation, or suspicious activities. Backup & Recovery System Including automatic backups of encrypted credentials will make sure that data is safe and reliable if file corruption takes place.

11. References

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
2. Deitel, P., & Deitel, H. (2016). *C: How to Program* (8th ed.). Pearson Education.
3. Tanenbaum, A. S. (2012). *Modern Operating Systems* (4th ed.). Pearson.
4. Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson Education.
5. Forouzan, B. A. (2007). *Cryptography and Network Security*. McGraw-Hill.
6. Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons.
7. Godse, A., & Godse, D. (2010). *Computer Networks*. Technical Publications.
8. Mandal, A. (2014). "Basic Encryption Techniques in C Programming." *International Journal of Computer Applications*, 102(15), 12–16.
9. GeeksforGeeks. (2023). "String Handling in C – Complete Guide." Available online at: www.geeksforgeeks.org
10. Tutorialspoint. (2023). "C Programming – Functions." Available online at: www.tutorialspoint.com
11. MDN Web Docs. (2023). "XOR Operation in Computing." Accessible at: developer.mozilla.org
12. NIST (2015). *Security Requirements for Cryptographic Modules*. National Institute of Standards and Technology.
13. CSE Labs (2022). "Simple Encryption and Decryption Algorithms." *CSE Technical Reports*.
14. ISO/IEC 27002 (2013). *Information Security Controls and Guidelines*. International Organization for Standardization.

12. Declaration

I understand that Karunya Institute of Technology and Sciences retains the rights to any product developed *(software or hardware)* through the Micro Project submitted by me to the Institution.

Furthermore, if I intend to document or publish any findings from the Micro Project work for commercial purposes, I shall obtain a No Objection Certificate (NOC) from the Office of the Registrar before engaging in such activities.

I also understand that any patentable outcomes arising from the Micro Project shall be reported to the Institution, and that patent filing, ownership, and revenue-sharing will be governed by the Intellectual Property Rights (IPR) policies of Karunya Institute of Technology and Sciences. I acknowledge that any commercial utilization or republication of the Micro Project work, in full or in part, shall require prior permission from Karunya Institute of Technology and Sciences. I further agree that any royalty accruing from such utilization will be shared between the inventor(s) and the Institution in the ratio of 60:40, respectively, in accordance with the Institute's Intellectual Property Rights policy

Signature of the Student:

Ronel

Name of the Student: RONEL ABRAHAM MATHEW

Register Number: URK25CS1106

Date: 22/11/2025

12. Declaration

I understand that Karunya Institute of Technology and Sciences retains the rights to any product developed *(software or hardware)* through the Micro Project submitted by me to the Institution.

Furthermore, if I intend to document or publish any findings from the Micro Project work for commercial purposes, I shall obtain a No Objection Certificate (NOC) from the Office of the Registrar before engaging in such activities.

I also understand that any patentable outcomes arising from the Micro Project shall be reported to the Institution, and that patent filing, ownership, and revenue-sharing will be governed by the Intellectual Property Rights (IPR) policies of Karunya Institute of Technology and Sciences. I acknowledge that any commercial utilization or republication of the Micro Project work, in full or in part, shall require prior permission from Karunya Institute of Technology and Sciences. I further agree that any royalty accruing from such utilization will be shared between the inventor(s) and the Institution in the ratio of 60:40, respectively, in accordance with the Institute's Intellectual Property Rights policy

Signature of the Student:

[illegible]

Name of the Student: JOSHUA SIBICHAN SCARIYA

Register Number: URK25CS1106

Date: 22/11/2025

12. Declaration

I understand that Karunya Institute of Technology and Sciences retains the rights to any product developed *(software or hardware)* through the Micro Project submitted by me to the Institution.

Furthermore, if I intend to document or publish any findings from the Micro Project work for commercial purposes, I shall obtain a No Objection Certificate (NOC) from the Office of the Registrar before engaging in such activities.

I also understand that any patentable outcomes arising from the Micro Project shall be reported to the Institution, and that patent filing, ownership, and revenue-sharing will be governed by the Intellectual Property Rights (IPR) policies of Karunya Institute of Technology and Sciences. I acknowledge that any commercial utilization or republication of the Micro Project work, in full or in part, shall require prior permission from Karunya Institute of Technology and Sciences. I further agree that any royalty accruing from such utilization will be shared between the inventor(s) and the Institution in the ratio of 60:40, respectively, in accordance with the Institute's Intellectual Property Rights policy

Signature of the Student:

[illegible]

Name of the Student: ADITI LAKSHMANAN

Register Number: URK25CS1106

Date: 22/11/2025

12. Declaration

I understand that Karunya Institute of Technology and Sciences retains the rights to any product developed *(software or hardware)* through the Micro Project submitted by me to the Institution.

Furthermore, if I intend to document or publish any findings from the Micro Project work for commercial purposes, I shall obtain a No Objection Certificate (NOC) from the Office of the Registrar before engaging in such activities.

I also understand that any patentable outcomes arising from the Micro Project shall be reported to the Institution, and that patent filing, ownership, and revenue-sharing will be governed by the Intellectual Property Rights (IPR) policies of Karunya Institute of Technology and Sciences. I acknowledge that any commercial utilization or republication of the Micro Project work, in full or in part, shall require prior permission from Karunya Institute of Technology and Sciences. I further agree that any royalty accruing from such utilization will be shared between the inventor(s) and the Institution in the ratio of 60:40, respectively, in accordance with the Institute's Intellectual Property Rights policy

Signature of the Student:

SD

Name of the Student: YOGESH S

Register Number: URK25CS1106

Date: 22/11/2025