CodeGen

PassGuard

Local Password Encryption & Management

OUR TEAM

JOSHUA

ADITI LAKSHMANAN

RONEL

YOGESH

PROJECT OBJECTIVE

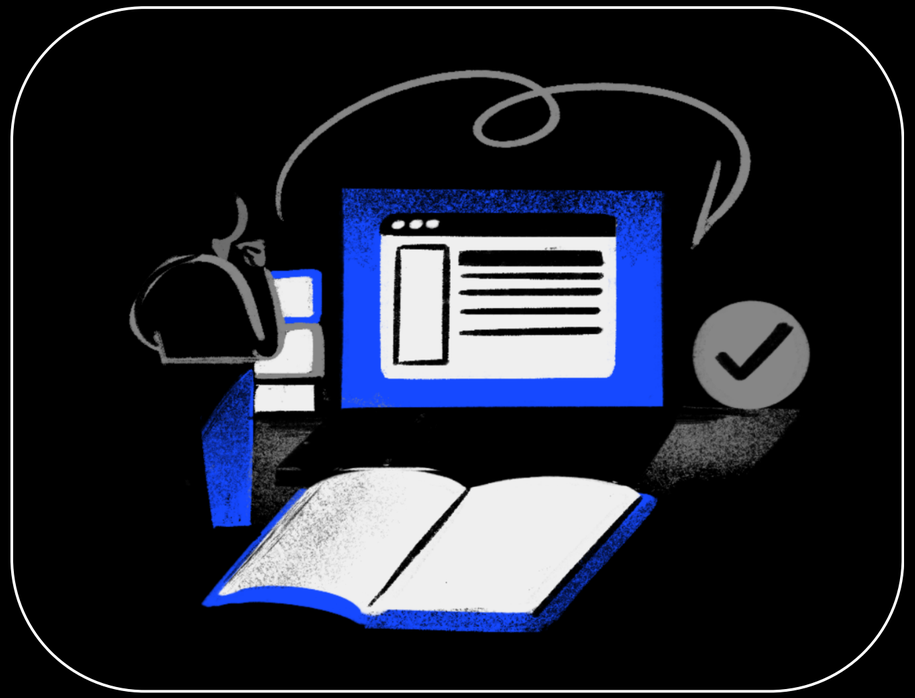
Secure password manager built in C with encryption and file storage.

GOALS

1. Implement password encryption using Caesar Cipher algorithm.
2. Develop a secure user authentication system.
3. Create persistent data storage with file I/O.
4. Build an interactive GUI using raylib library.
5. Apply C programming concepts: structs, arrays, string handling, functions.

Key Concept: Passwords are encrypted before storage and decrypted only during login verification

WHY PASSWORD ENCRYPTION MATTERS



Passwords are stored in encrypted form to:

1. **Prevent unauthorized access** - Even if database is breached, passwords remain protected
2. **Ensure data privacy** - Users' credentials cannot be read by attackers
3. **Comply with security standards** - Industry best practice for credential storage
4. **Protect user accounts** - Encryption adds a security layer beyond access controls

Real-world Impact: Big companies have lost millions of user accounts from weak encryption.

Our Solution: Caesar Cipher encryption (Key: 5) for educational demonstration.

CAESAR CIPHER ENCRYPTION EXPLAINED

Caesar Cipher shifts each character by a fixed number (key = 5 in our project):

Encryption Formula:

Encrypted_Char = (Original_Char + Key) mod 26

Example Transition:

Original : PASSWORD

Encrypted : UFXXBTWI

ENCRYPTION & DECRYPTION PROCESS

Registration Process (Encryption):

```
void caesarEncrypt(char *password, char *encrypted, int key) {
    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';
        }
    }
```

Login Process (Decryption):

```
void caesarDecrypt(char *encrypted, char *decrypted, int key) {
    // Reverse the process: subtract key instead of adding
    decrypted[i] = ((encrypted[i] - 'A' - key + 26) % 26) + 'A';
}
```

Flow: User Password → Encrypt → Store in Database → Login → Decrypt → Verify

DATA STRUCTURES & USER STORAGE

User Structure Definition:

```
typedef struct {
    char username[50];
    char
encrypted_password[MAX_PASSWORD_LE
NGTH];
    int is_active;
} User;

User users[MAX_USERS];  // Array to store
up to 10 users
int user_count = 0;
```

Storage System:

1. **In-Memory:** Users stored in array during runtime
2. **File Persistence:** Save to 'users.dat' file
3. **Data Format:** Tab-separated values (username | encrypted_password | is_active)

Security Features

1. Minimum password length: 6 characters
2. No spaces allowed in username/password
3. Unique username validation
4. Linear search for user lookup (O(n) complexity)

PROJECT FEATURE AND FUNCTIONALITY

User Registration

Allows new users to create accounts

View Users

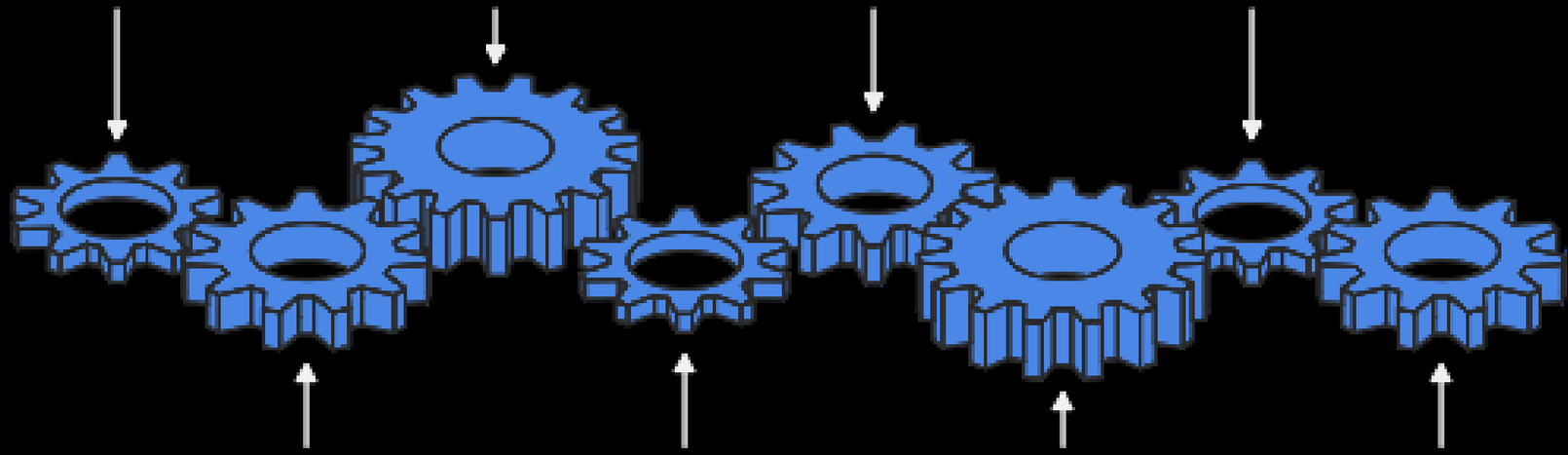
Displays a list of all users

Sort Users

Arranges users in a specific order

Load Data

Retrieves previously saved user data



Login

Enables users to access their accounts

Search User

Finds specific users based on criteria

Save Data

Stores current user data

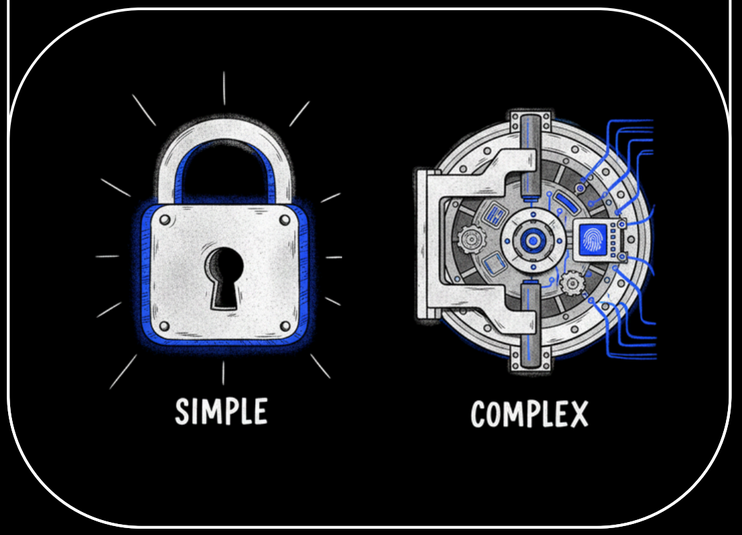
Exit

Closes the application

OUR UNIQUE VALUE

Why Our Project Stands Out:

1. **Transparent Code**
   All encryption logic visible and understandable
2. **Educational First**
   Learn how password security actually works
3. **Minimal Dependencies**
   Pure C + raylib only (no bloat)
4. **Complete Control**
   Local storage, user owns their data
5. **Live Demo**
   See encryption/decryption happening in real-time
6. **From Scratch**
   Not a wrapper around existing frameworks



USER INTERFACE & SCREENS

Screen Modules

1. Main Menu Screen

- 8 interactive buttons for all operations
- Displays total user count and encryption key
- Color-coded buttons (Green, Blue, Purple, Orange, Red, etc.)

2. Registration Screen

- Username input field
- Password input field (masked display with *)
- Password validation: min 6 characters, no spaces

3. Login Screen

- Username and password input fields
- Decryption and verification logic
- Success/Error message feedback

4. View User Screen

- Table display: No. | Username | Encrypted Password | Hash Value
- Shows hash of decrypted password

5. Search Screen

- Search field for username lookup
- Returns user existence status

Technology: Built with Raylib (Graphics Library)

- Window: 1000 x 700 pixels
- 60 FPS rendering
- Interactive button collision detection

ALGORITHM: SEARCH & SORT

1. Linear Search - User Lookup (O(n))

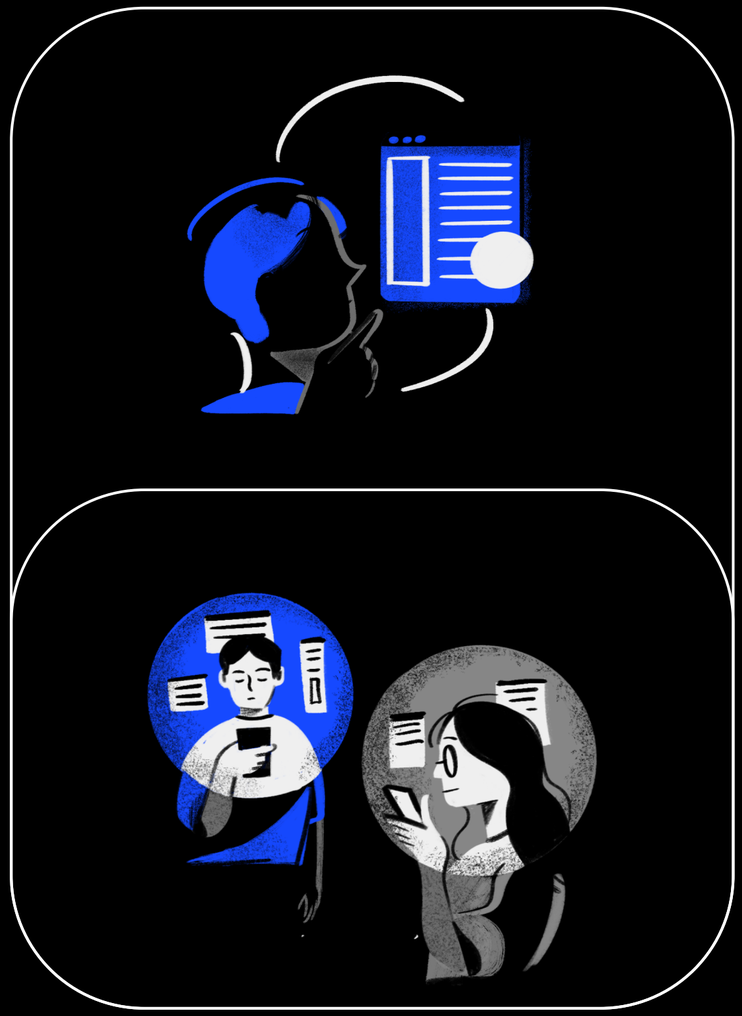
Searches through user array to find matching username. Used for registration check, login verification, and search operations.

2. Bubble Sort - Alphabetical User Sorting ($O(n^2)$)

Compares adjacent usernames and swaps them to arrange in alphabetical order. Used to organize users list before display.

3. Hash Function - Password Validation

Generates unique numeric hash value from password for verification purposes. Helps validate password integrity.



Complexity Analysis:

- Linear Search: Fast for small datasets (≤100 users)
- Bubble Sort: Simple but slower for larger lists
- Hash: Constant time O(1) for generation"

FILE I/O & DATA PERSISTENCE

1. Save to File Operation

Writes all registered users to 'users.dat' file for permanent storage. Stores username, encrypted password, and user status.

2. Load from File Operation

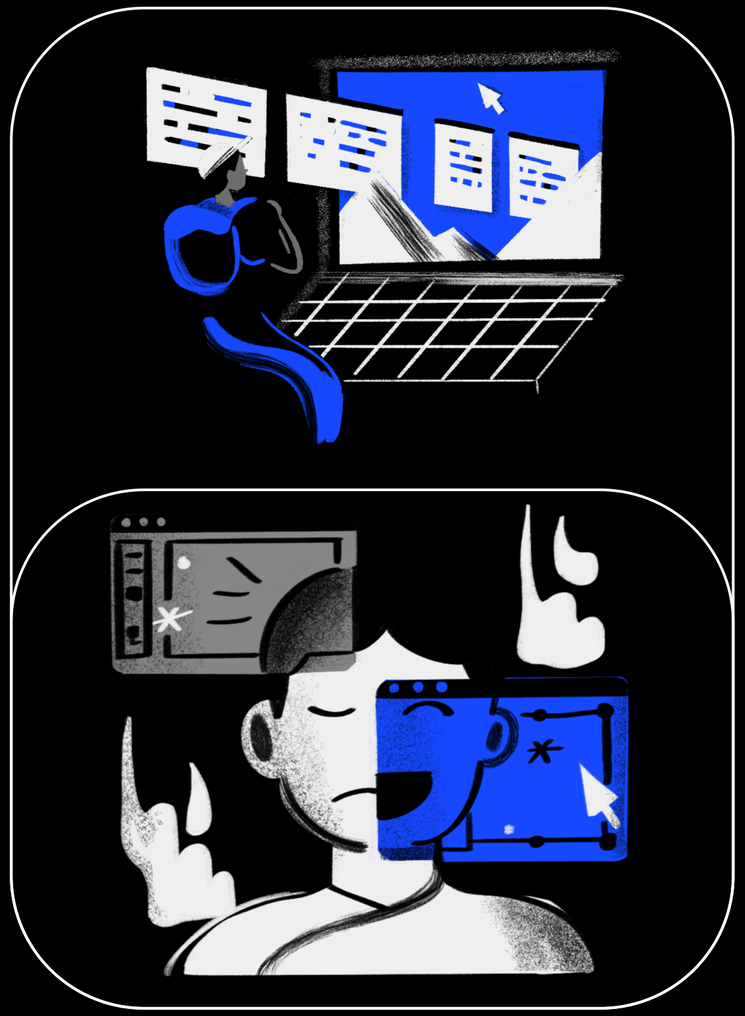
Reads user data from 'users.dat' file and populates the application with previously saved users on startup.

3. Hash Function - Password Validation

3
alice ufxxbtwi 1
bob yjkkvtmj 1
charlie hsvvc 1

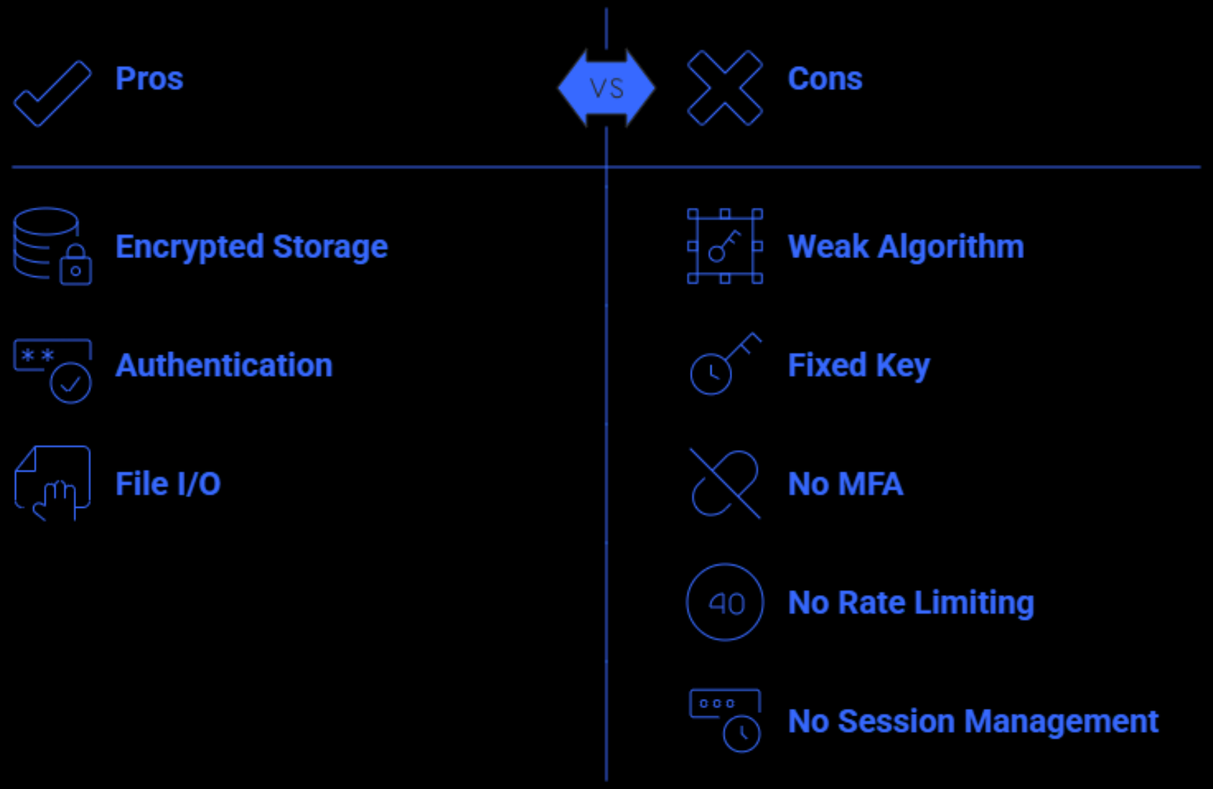
Benefits:

- Data persists after application closes
- Passwords remain encrypted in storage
- Easy to backup or share user database
- Simple text format for easy debugging



SECURITY ANALYSIS & LIMITATIONS

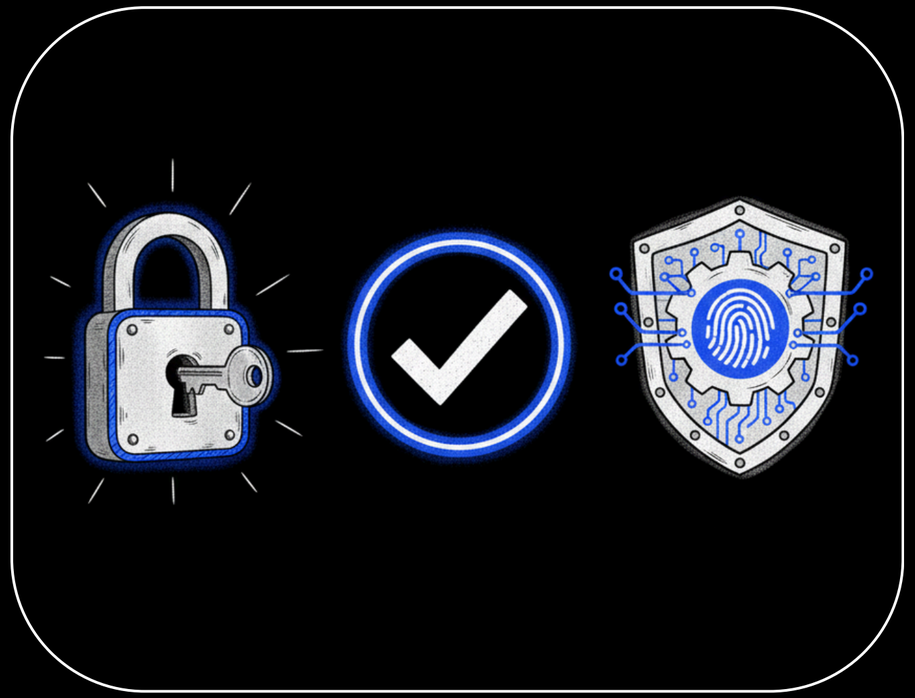
Caesar Cipher vs. Industry Standard



SOURCE CODE

Github Repository

CONCLUSION & KEY TAKEAWAYS



What We Built:

A complete password management system demonstrating core cybersecurity principles using C programming.

Key Learning:

1. **Caesar Cipher encryption fundamentals**
2. **User authentication and credential management**
3. **Data structures (structs, arrays) for user storage**
4. **File I/O for persistent data management**
5. **GUI development with raylib library**
6. **Algorithm implementation (search, sort)**
7. **Security best practices and limitations**

THANK YOU