(Group-7) Secure Personal Diary Management System

Fabiha Raiyan
2421170642

Aminur Rashid Sohag
2514160042

Tanvir Ahmed Ovi
2512723642

Imtiaj Ahmed
2513119042

Abstract— This paper presents the development of a secure Personal Diary Management System using the C programming language. The system provides features such as encrypted storage, date validation, PIN authentication, and a structured interface for managing personal diary entries. The primary motivation behind this project is to offer a simple yet secure way for individuals to digitally document daily experiences, reflections, or notes, while ensuring privacy. This system uses XOR-based encryption for securing user data and file handling techniques for storing diary entries persistently. This report details the design, implementation, testing, and future enhancement scope of the application. The overall goal is to bridge the gap between simplicity and security using a low-level language like C.

This diary system project acts as a practical demonstration of the power of procedural programming in achieving personal data security and highlights the C language's capability in handling real-life applications. It ensures a safe personal journaling experience for the user, especially in environments where cloud-based apps are not desirable or feasible.

I. INTRODUCTION

Digital journaling has gained popularity as a form of self-expression, therapy, and recordkeeping. However, cloud-based journaling apps raise concerns regarding user privacy and data security. Users often hesitate to write personal thoughts on online platforms due to potential data breaches, unauthorized access, or third-party data tracking. A locally installed, offline digital diary system resolves these issues by offering privacy control and independence from internet reliance.

The project described in this paper implements such a secure offline diary system using the C programming language. C, being a low-level procedural language, gives the programmer complete control over memory and data management, making it suitable for building performance-critical and secure applications. The diary system allows users to write, store, edit, and manage personal notes, with encryption mechanisms to protect sensitive content.

We wanted to:

- Develop a secure local application for diary management.
- Introduce file-level encryption.
- Implement intuitive features accessible via command-line.
- Validate input data for accuracy and structure.
- Explore modularity and reusability in C.

This paper outlines the software's motivation, architecture, development methodology, core codebase, test outcomes, and proposed extensions, and it reflects on how classic C language practices can contribute to modern data privacy needs.

II. SYSTEM OBJECTIVES

System Objectives

The key objectives of the diary system include:

- **Confidentiality**: Ensure that diary entries are accessible only to the authorized user.
- **Portability**: Build a lightweight, local application that can run on any C-compatible system.
- **Data Integrity**: Validate all user inputs to avoid malformed or corrupted entries.
- **Security**: Use a basic encryption technique to protect stored data.

- **User Experience**: Provide a clear, menu-driven interface for easy operation.
- **Maintainability**: Use modular programming for easier extension and debugging.
- **Offline Operation**: Function fully without internet dependency.
- **Educational Value**: Serve as a project-based learning tool for students of C.

III. METHODOLODY

The application was developed using the Waterfall Model, following these steps:

- **Requirements Gathering**: Identifying system goals, target users, and operational needs.
- **System Design**: The system structure was planned using pseudocode and module charts. The architecture was broken into logical blocks like authentication, input validation, and file handling.
- **Implementation**: Writing the C code using Code::Blocks and GCC.
- **Testing**: Running unit tests and manual validations across features.
- **Documentation**: Preparing user guides and academic documentation.
- **Review and Iteration**: Refining the code based on test feedback.

Each function of the system corresponds to one module, promoting separation of concerns and code reuse.

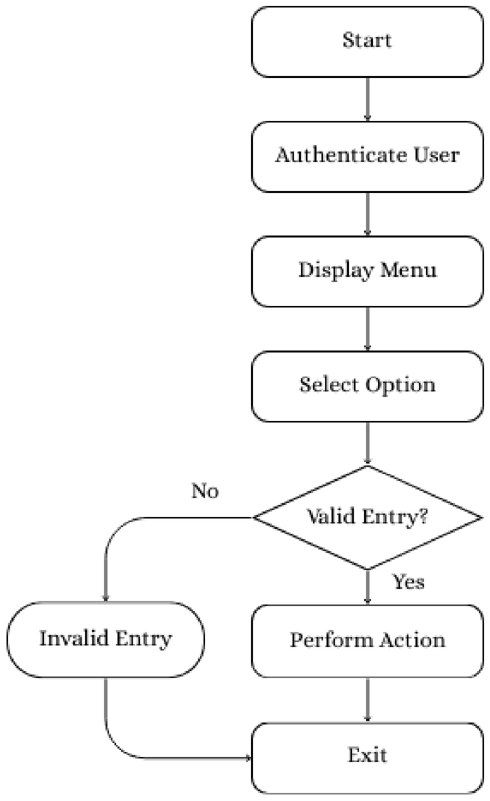
IV. ARCHITECTURE

4.1 System Overview

The system operates in five major stages:

1. **User Authentication**: Requires the user to input a valid PIN before access.
2. **Loading Entries**: Reads the diary file and decrypts existing entries.
3. **Menu Interaction**: Displays options and navigates user input.
4. **Entry Operations**: Executes add, edit, delete, view, or search functions.
5. **Saving Entries**: Encrypts and writes entries back to the binary file.

4.2 Flowchart



V. SYSTEM DESIGN

The system consists of several modules:

- **Authentication Module**: Secures access using a PIN code.
- **Input Validation Module**: Ensures correct date formatting and logic.
- **Encryption/Decryption Module**: Provides XOR-based text encryption.
- **File Storage Module**: Handles saving and retrieving diary entries in binary mode.
- **CRUD Operations Module**: Enables Add, View, Search, Edit, and Delete functionalities.

The overall program architecture is modular and follows a top-down design. Each function is defined separately, and a global structure is used to hold diary entries in memory. User interactions are managed through a simple menu interface that loops until the user exits.

The system begins with user authentication. After a successful login, it loads any existing diary entries into memory. The user can then add, view, search, edit, or delete entries. Each operation is immediately saved to disk after execution to ensure data persistence.

VI. FEATURES AND IMPLEMENTATION

The application is implemented in C using Code::Blocks IDE and standard C libraries. It supports the following features:

- **PIN Authentication**: Upon launching, the system prompts the user to enter a PIN. Access is denied if the PIN is incorrect. This enhances personal security and protects against unauthorized access. Although simple, it ensures that unauthorized users cannot access diary entries unless they know the PIN. This method prevents accidental access on shared machines and adds a basic layer of protection.
- **Date Validation**: Entries require a valid date in the format YYYY-MM-DD. The system uses `sscanf` to parse the input string into year, month, and day integers. It then checks for validity, including leap year logic for February. This prevents logical errors in recordkeeping. If an invalid date is entered, the program outputs an appropriate message and cancels the current operation. This protects the integrity of the entry archive.
- **Add Entry** Upon selection, the user is prompted for a date, a title, and the diary content. The system first validates the date, checks for duplicates, and if valid, stores the entry in an array of `DiaryEntry` structs. The entry is then encrypted and saved to the binary file. The system ensures each date only has one corresponding entry. If an entry for the date already exists, the user is instructed to use the edit function.
- **View Entries**: Entries are read from the binary file during startup and stored in memory. When the user selects the "View Entries" option, all entries are decrypted and displayed in chronological order of entry (not date-sorted).
- **Search Entry**: The search function prompts the user to enter a date in the YYYY-MM-DD format. It then compares the input string with the date fields of all loaded entries. If a match is found, the associated title and content are displayed. If no match is found, the user is notified that the entry does not exist. The date entered is also validated before performing the search.
- **Edit Entry** The system first prompts the user to enter a valid date. If an entry exists for that date, it displays the current title and content, then prompts for new input. The entry is updated in memory and the entire array is re-saved to the file after re-encryption.
- **Delete Entry**: The user is prompted to enter a date. If an entry is found for the given date, it is displayed and the user is asked to confirm deletion. Upon confirmation, the entry is removed by shifting the array and decrementing the entry count. All entries are then re-saved.
- **Encryption**: XOR cipher is applied to each field before saving and after reading from the file. It ensures file contents remain unreadable to outside users. This type of encryption, while simple, demonstrates the concept of symmetric encryption.

  The data is stored in a binary file named `diaryy.txt`. The use of binary format adds an additional layer of obfuscation compared to plain text files.

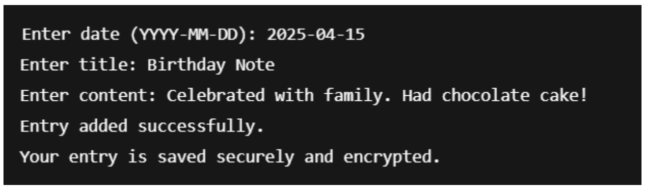
  Advantages of XOR:

  - Lightweight
  - Fast on low-end hardware
  - Simple to implement

  Limitations:

  - Vulnerable if key is discovered

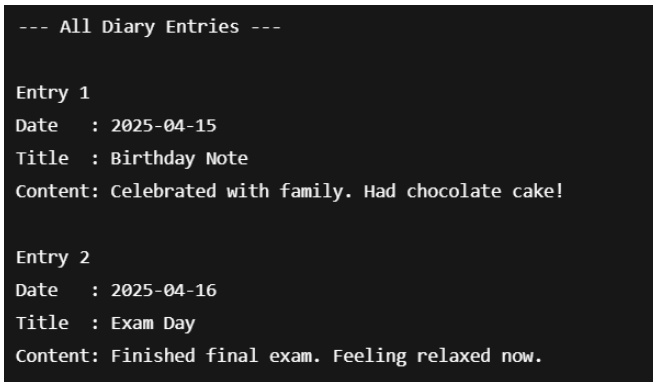
Figure 1. Add Entry:



```
Enter date (YYYY-MM-DD): 2025-04-15
Enter title: Birthday Note
Enter content: Celebrated with family. Had chocolate cake!
Entry added successfully.
Your entry is saved securely and encrypted.
```

z

Figure 2. View Entries:

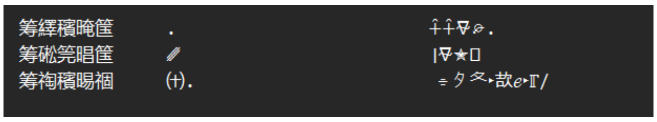


```
--- All Diary Entries ---

Entry 1
Date   : 2025-04-15
Title  : Birthday Note
Content: Celebrated with family. Had chocolate cake!

Entry 2
Date   : 2025-04-16
Title  : Exam Day
Content: Finished final exam. Feeling relaxed now.
```

Figure 3. Encrypted File Example:



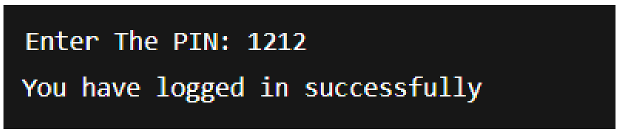
```
筹繹穧晻筐      .                        ̂+̂+∇ø.
筹磁笎唱筐      ⫽                        |∇★□
筹祹穧暘裀      (+).                      ≑タ⼡‣款e‣Ⅱ/
```

The interface is console-based, with clear instructions to the user at every step. Input handling includes newline stripping and buffer clearing to ensure no residual input affects execution.

VII. CODE EXPLANATION

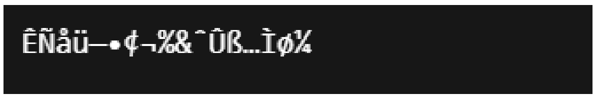
The code is modular and includes the following key functions:

- authenticate(): Validates a 4-digit PIN.



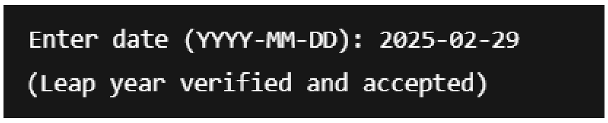
```
Enter The PIN: 1212
You have logged in successfully
```

- xorEncryptDecrypt(char *text): Applies XOR encryption with a static key. Even if a user opens the `diaryy.txt` file with a text editor, they will only see unreadable binary data, thanks to the encryption.



```
ÊÑåü—•¢¬%&ˆÛß…Ìø¼
```

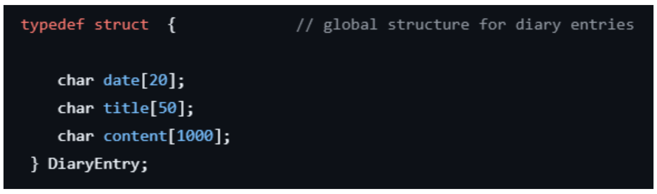
- isValidDate(y, m, d): Validates logical correctness of the date.



```
Enter date (YYYY-MM-DD): 2025-02-29
(Leap year verified and accepted)
```

- addEntry(), viewEntry(), searchEntry(), editEntry(), and deleteEntry() handle respective CRUD operations.
- saveEntries() and loadEntries() manage persistent storage using binary file I/O.

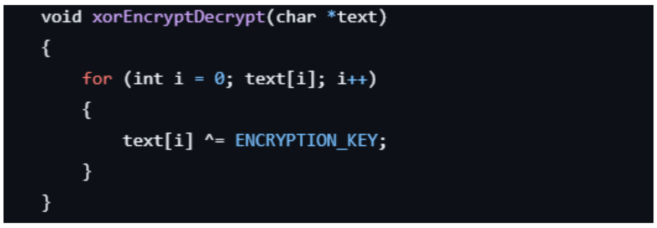
The diary entries are stored in a structure:



```c
typedef struct  {              // global structure for diary entries

    char date[20];
    char title[50];
    char content[1000];
 } DiaryEntry;
```

Sample Code Snippet For X-OR Encryption:



```c
void xorEncryptDecrypt(char *text)
{
    for (int i = 0; text[i]; i++)
    {
        text[i] ^= ENCRYPTION_KEY;
    }
}
```

This structure supports all the diary operations. Entries are stored in an array. When the application starts, existing entries are loaded into this array and decrypted. New entries are encrypted before being stored.

Each action uses standard file handling (fopen, fwrite, fread, fclose) in binary mode to prevent format issues and ensure performance. The use of binary files ensures efficiency and prevents easy tampering of saved records.

Error checking is incorporated to ensure robust file operations. If any file operation fails, the program gracefully notifies the user.

VIII. TESTING AND VERIFICATION

Rigorous testing was carried out using both valid and invalid inputs:

- **Boundary Testing:** Ensured date logic passed edge cases such as leap years.
- **Encryption Validation:** Verified file content was unreadable post-save.
- **Persistence Check:** Confirmed data retained across sessions.
- **Invalid Input Handling:** Tested response to bad dates, wrong PINs, and duplicate entries.

Test Matrix:

Test Case	**Input**	**Expected Output**	**Result**
Valid Pin	1212	Success	Pass
Invalid Pin	0000	Access Denied	Pass
Invalid Date	2025-13-40	Error	Pass
Duplicate Entry	Same Date	Rejected	Pass
View all	-	List Entries	Pass
Edit Entry	Valid Date	Updated	Pass
Delete Entry	Valid Date	Removed	Pass

IX. USER INTERFACE AND EXPERIENCE

The user interface is purely text-based. While simple, it is responsive and efficient. All inputs are line-buffered to avoid overflow, and instructions are printed clearly for each operation. The application includes error messages and success prompts, giving the user continuous feedback.

This design was chosen to keep the program lightweight and compatible across platforms without the need for graphic libraries.

X. USER MANUAL

Step-by-step Usage:

1. Run the compiled program.
2. Enter your secure PIN (default: `1212`).
3. Choose from menu:
    - Add Entry
    - View Entries
    - Search Entry
    - Edit Entry
    - Delete Entry
    - Exit
4. Follow on-screen instructions for each operation.
5. All entries are automatically saved to encrypted file.

File Location: `diaryy.txt`

XI. CHALLENGES FACED

During development, several key challenges were encountered:

- **Handling buffer overflows**: The standard C library functions such as `scanf` and `fgets` are notorious for their potential to cause buffer overflows if not used properly. Initially, using `%s` in `scanf` to read user input led to undefined behavior when users entered strings longer than expected, resulting in crashes or memory corruption.
- **Date validation:** for leap years and malformed inputs.
- **Encrypted binary file handling**: Making sure the file remains unreadable through normal editors yet remains readable by our decryption algorithm.
- **Ensuring Data Consistency Across CRUD Operations:** When entries were updated or deleted from the middle of the binary file. Any misalignment or failure to shift file contents correctly could result in lost or corrupted entries.

These challenges were resolved through:

- Modular function design
- Using length specifiers and replacing scanf with fgets wherever feasible. Additional input cleanups.
- Extensive testing
- Careful use of string functions.
- Implemented a temporary file strategy for edit and delete operations.

XII. LIMITATIONS AND SCOPES FOR IMPROVEMENT

12.1 Limitations

- Only one user is supported; no user profiles.
- No static PIN Authentication
- Basic X-OR Encryption method used, which is not suitable for security-critical applications.
- No data backup or export options.
- System is entirely text-based.
- Limited Entry Capacity.
- Entries are not displayed in chronological order based on dates.

12.2 Future Enhancements

- **Multi User Support:** Introduce user account creation, login, and logout functionality. Each user would have a separate diary file, personalized settings, and individual authentication credentials. *(structures with file names per user)*
- **Stronger Encryption** *(no idea yet.)*
- **Export Entries:** Export diary to `.txt`/`.pdf` *(`fopen()`, `fprintf()` for text; PDF libraries like **libharu** for `.pdf`)*
- **Search by Keywords:** Allow users to search diary entries using words *(`strstr()`, `strcmp()`)*
- **Unlimited Entries:** Allow users to input as many entries as they want. *(by replacing the fixed-size array with a dynamically allocated one using `malloc()` and `realloc()` to allow unlimited diary entries based on available memory.)*

XIII. EDUCATIONAL VALUE

This project serves as an excellent exercise for students learning C. It introduces:

- Struct usage and dynamic arrays
- File I/O in binary format
- String manipulation and validation
- Modular programming
- Basic encryption
- Real-world application design

XIV. CONCLUSION

The development of the Secure Personal Diary Management System in C highlights the practical application of core programming principles to solve real-world problems centered around privacy, user autonomy, and data persistence. In an age where most journaling applications are cloud-based and subscription-driven, this project presents a valuable alternative: a lightweight, offline, and secure personal diary system that offers complete control to the end-user.

Throughout development, the project addressed key areas such as file handling, input validation, basic encryption, and memory management, all within the constraints of the C programming language. The use of a modular codebase and structured programming techniques allowed the software to remain maintainable and adaptable for future upgrades. The system achieves its primary goals: protecting user data through encryption, validating input to prevent errors, enabling core CRUD operations, and persisting data across sessions. That said, there is significant room for future enhancement.

In essence, this project provides a strong foundation for building secure and user-centric applications and reinforces the relevance of offline tools in a cloud-dominated world.

XV. REFERENCES

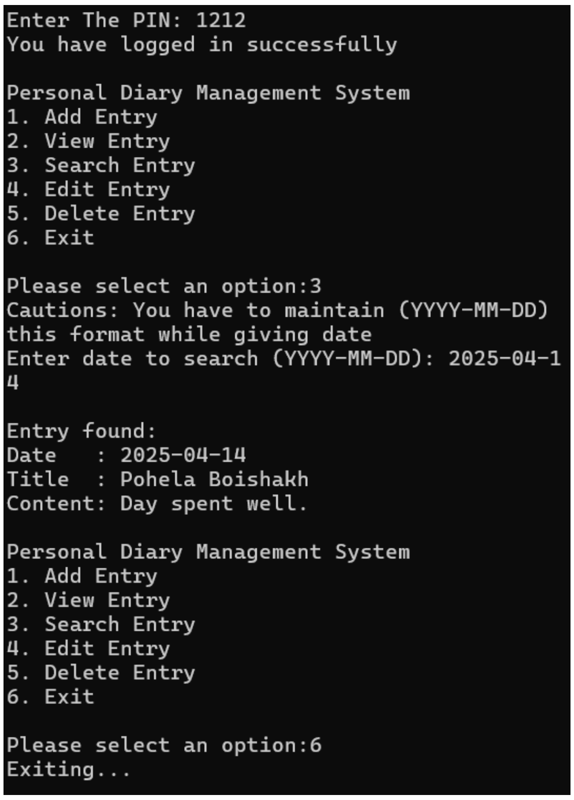
[1] University of Delhi, "Double Column Research Paper Format," Sep. 23, 2015. [Online]. Available: https://www.du.ac.in/du/uploads/events/23092015/23092015_Double%20Column%20Research%20Paper%20Format.pdf

[2] A. Caulfield, "IEEE Paper Format," Scribbr, [Online]. Available: https://www.scribbr.com/ieee/ieee-paper-format/

[3] GeeksforGeeks, "C Programming Language," [Online]. Available: https://www.geeksforgeeks.org/c-programming-language/

[4] W3Schools, "C Tutorial," [Online]. Available: https://www.w3schools.com/c/

XVI. **APPENDIX A:** EXTENDED OUTPUT SAMPLES



```
Enter The PIN: 1212
You have logged in successfully

Personal Diary Management System
1. Add Entry
2. View Entry
3. Search Entry
4. Edit Entry
5. Delete Entry
6. Exit

Please select an option:3
Cautions: You have to maintain (YYYY-MM-DD)
this format while giving date
Enter date to search (YYYY-MM-DD): 2025-04-1
4

Entry found:
Date    : 2025-04-14
Title   : Pohela Boishakh
Content: Day spent well.

Personal Diary Management System
1. Add Entry
2. View Entry
3. Search Entry
4. Edit Entry
5. Delete Entry
6. Exit

Please select an option:6
Exiting...
```

XVII. **APPENDIX B**: COMPLETE TEST CASES

Case	Description	Input	Expected Output
TC01	Valid PIN	1212	Logged in
TC02	Invalid PIN	0000	Access Denied
TC03	Add valid entry	Date, title, content	Entry saved
TC04	Duplicate Date	Existing date	Error: entry exists
TC05	View Entries(1+ exist)	-	Entry list
TC06	View Entries(none exist)	-	No entries to display
TC07	Edit existing entry	Existing date, New data	Entry updated
TC08	Edit non existent date	Invalid Date	Entry not found

TC09	Delete entry	Valid Date + confirm ‘y’	Entry deleted
TC10	Delete Cancelled	Valid Date + confirm ‘n’	Deletion cancelled.

XVIII. **APPENDIX C:** PARTIAL CODE WITH COMMENTS

```
    // Ask user to enter a valid date
    do {
        printf("Enter date (YYYY-MM-DD): "); // Prompt user to enter a date in YYYY-MM-DD format
        fgets(dateInput, sizeof(dateInput), stdin); // Read date input as a string
        dateInput[strcspn(dateInput, "\n")] = 0; //remove newline
        if (sscanf(dateInput, "%d-%d-%d", &y, &m, &d) != 3 || !isValidDate(y, m, d)) {
            printf("Invalid date! Try again.\n");
        } else {
            break; // Break out of the loop once a valid date is entered
  }
    } while (1); // Keep asking until a valid date is entered
    // Loop through entries to find a matching date
    for (int i = 0; i < entryCount; i++) {
        if (strcmp(entries[i].date, dateInput) == 0) {
            found = 1;
            printf("Current Title: %s\n", entries[i].title);
            printf("Current Content: %s\n", entries[i].content);

            printf("Enter new title: ");
            fgets(entries[i].title, sizeof(entries[i].title), stdin);
            entries[i].title[strcspn(entries[i].title, "\n")] = 0;
            printf("Enter new content: ");
            fgets(entries[i].content, sizeof(entries[i].content), stdin);
            entries[i].content[strcspn(entries[i].content, "\n")] = 0;
            saveEntries(); // save updated entries
            printf("Entry updated successfully.\n");
            break;
        }
    }
    // If no entry matching the date was found, notify the user
    if (!found) {
        printf("No entry found for the given date.\n");
    }
}
```