ENCRYPTED CLOUD FILE STORAGE SYSTEM

Project Report

Course: CSEG1032 - Programming in C
University: University of Petroleum and Energy Studies
Student name: Md Ejaz
Instructor: Dr. Tanu Singh

TABLE OF CONTENTS

---

1. ABSTRACT

This project presents an Encrypted Cloud File Storage System developed in C programming language. The system provides secure file storage capabilities with user authentication, file encryption, and complete file management operations. The application uses XOR cipher for encryption and implements a metadata-based file tracking system. Each user has isolated storage space with password-protected

access, ensuring data privacy and security. The system demonstrates practical implementation of file handling, encryption algorithms, dynamic memory management, and modular programming in C.

Keywords: Cloud Storage, File Encryption, XOR Cipher, User Authentication, File Management, C Programming

---

2. PROBLEM DEFINITION

2.1 Background

In today's digital world, secure file storage is essential. Users need systems that can:

- Store files securely with encryption
- Provide user authentication and access control
- Manage multiple files efficiently
- Ensure data privacy between different users

2.2 Problem Statement

Develop a command-line based encrypted cloud file storage system that allows multiple users to securely upload, download, delete, and manage their files with the following requirements:

1. **User Management**: Secure login and account creation
2. **File Encryption**: Encrypt files before storage
3. **File Operations**: Upload, download, delete, and list files
4. **Security**: Password protection and key-based decryption
5. **User Isolation**: Separate storage for each user

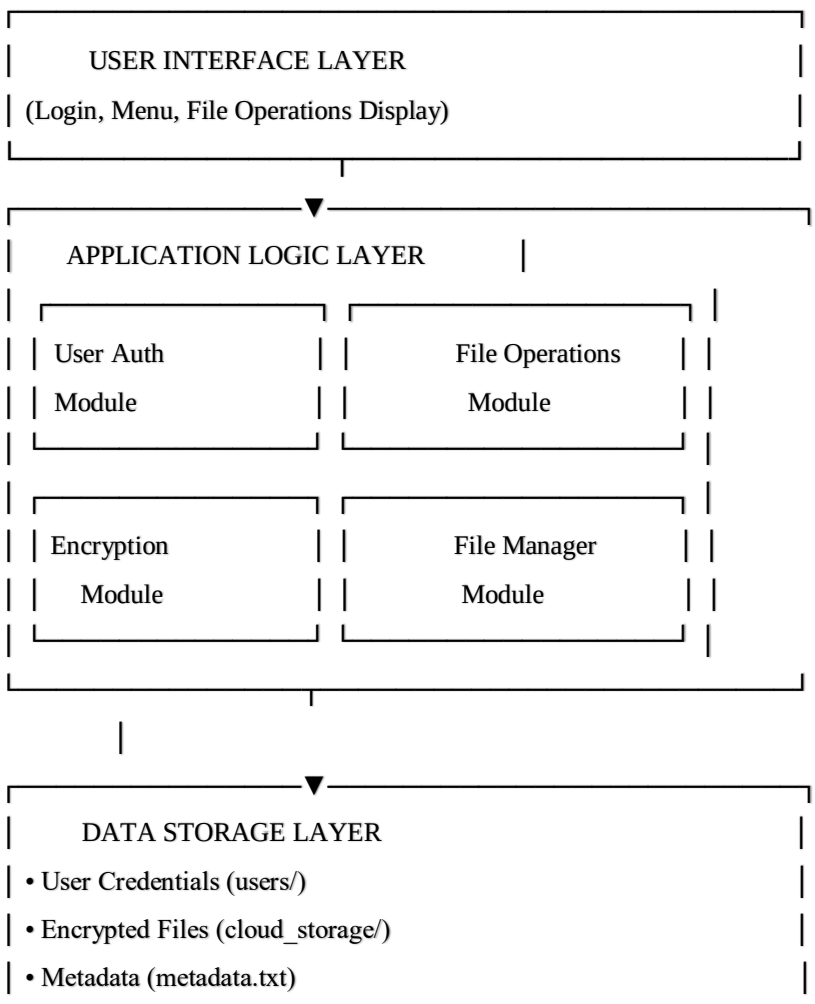
2.3 Objectives

- Implement secure user authentication system
- Develop file encryption/decryption using XOR cipher
- Create metadata management for file tracking
- Build modular code structure with header files
- Ensure cross-platform compatibility (Windows/Linux)
- Provide user-friendly command-line interface

---

3. SYSTEM DESIGN

3.1 System Architecture



3.2 Module Design

Module 1: Cloud Operations (cloud_operations.c/h)

- User login and account creation
- File upload with encryption
- File download with decryption
- File deletion with password verification
- File listing

Module 2: Encryption (encryption.c/h)

- XOR encryption algorithm
- XOR decryption algorithm

Module 3: File Manager (file_manager.c/h)

- Unique file ID generation
- Metadata creation and management
- Key hash generation
- File search functionality

Module 4: Main Program (main.c)

- Program initialization
- Menu-driven interface
- Function orchestration

3.3 Data Flow

Diagram Upload

Process:

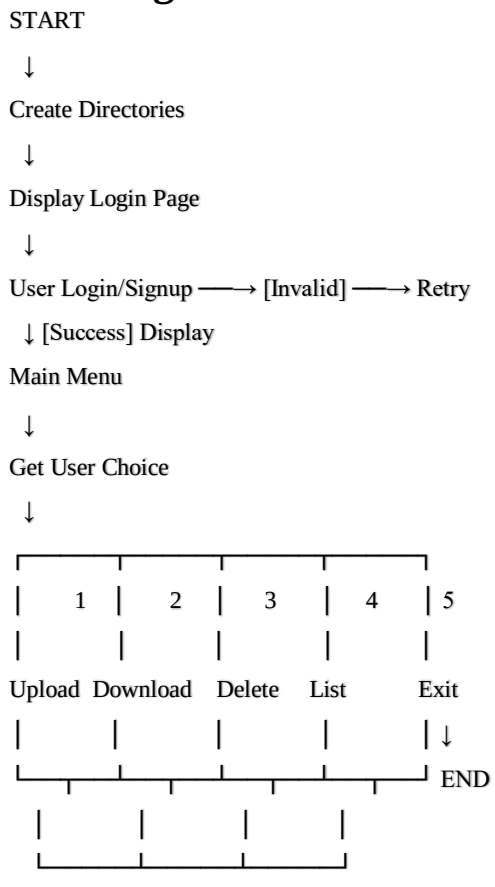
User Input → Read File → Encrypt Data → Generate File ID
→ Save Encrypted File → Update Metadata → Confirm Success

Download Process:

User Input (File ID) → Search Metadata → Verify Key
→ Read Encrypted File → Decrypt Data → Save to Output File

3.4 Flowcharts

Main Program Flow:



Upload File Algorithm:

1. START Upload
2. Input: filename, encryption_key
3. Open file in binary read mode
4. IF file not
   found THEN
   Display error
   RETURN
5. Generate unique file_id using timestamp
6. Read entire file into buffer
7. Encrypt buffer using XOR with key
8. Save encrypted data to cloud_storage/username/
9. Generate key hash
10. Add entry to metadata.txt
11. Display file_id to user
12. END

Download File Algorithm:

1. START Download
2. Input: file_id, decryption_key
3. Search metadata for file_id
4. IF not
   found
   THEN
   Display
   error
   RETURN
5. Get stored key hash from metadata
6. Generate hash of input key

7. IF hashes don't
   match THEN
   Display "Wrong
   key" RETURN
8. Read encrypted file into buffer
9. Decrypt buffer using XOR
10. Input: output_filename
11. Write decrypted data to output file
12. Display success message
13. END

3.5 Database Design

User Credentials File (users/username.txt):

Format:

password_hash

Example: 2090756197

Metadata File (cloud_storage/username/metadata.txt):

Format:

FILE_ID|original_name|version|key_hash

Example:

FILE_1733327845|test.txt|v1|2090756197

---

4. IMPLEMENTATION DETAILS

4.1 Key Data Structures

```
// Global user state
static char current username[50];
static unsigned int current_password_hash;

// File operations use dynamic memory
char *buffer = (char*)malloc(file_size);
```

4.2 Core Functions

4.2.1 User

Authentication

Account Creation**:**

```
static void create_account() {
    // Read username and password from user
    // Check if the user already exists
```

```
    // Generate password hash using DJB2 algorithm
    // Store the hash inside users/username.txt
    // Create a user-specific cloud storage folder
}
```

Login Verification:

```
static int login_existing_user() {
    // Read username and password from user
    // Load stored hash from the user's file
    // Compare the entered password hash with the stored hash
    // If matched, set current user session
    // Return success or failure
}
```

4.2.2 Encryption

Implementation XOR

Cipher:

```
void xor_encrypt(char *data, int length, char *key) {

    int key_len = strlen(key);

    for (int i = 0; i < length; i++) {

        data[i] = data[i] ^ key[i % key_len];

    }
```

Key Features:

- Symmetric encryption (same function for encrypt/decrypt)
- Key repetition for files longer than key
- Byte-by-byte XOR operation

4.2.3 File ID Generation

```
void xor_encrypt(char *data, int length, char *key) {
    int key_len = strlen(key);

    for (int i = 0; i < length; i++) {
        data[i] = data[i] ^ key[i % key_len];
    }
}
```

Uses Unix timestamp to ensure uniqueness.

4.2.4 Hash Function (DJB2)

```
unsigned int generate_key_hash(char *key) {
    unsigned int hash = 5381;   // Common DJB2 initial value
    int c;

    while ((c = *key++)) {
        hash = ((hash << 5) + hash) + c;  // hash * 33 + character
    }

    return hash;
}
```

4.3 File Management

Metadata Management:

- Each user has separate metadata.txt file
- Format: FILE_ID|original_name|version|key_hash
- Parsed using sscanf for structured data extraction

File Operations:

- Binary file handling for universal file type support
- Dynamic memory allocation for variable file sizes
- Proper resource cleanup (fclose, free)

4.4 Cross-Platform Compatibility

```
unsigned int generate_key_hash(char *key) {
    unsigned int hash = 5381;   // DJB2 starting value
    int c;

    while ((c = *key++)) {
        hash = ((hash << 5) + hash) + c;
        // hash * 33 + c
    }

    return hash;
}
```

4.5 Error Handling

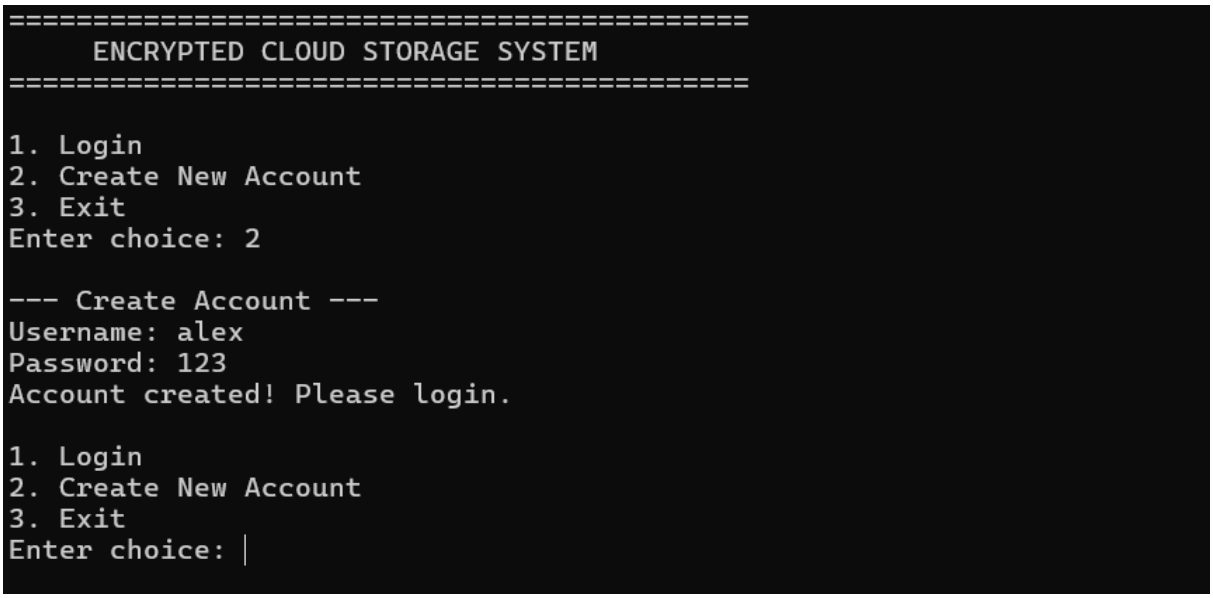
- File existence validation before operations
- Password verification for critical operations
- Memory allocation checks
- Invalid input handling in menu system

---

5. TESTING & RESULTS

5.1 Test Cases

Test Case 1: User Registration



Test Case 2: User Login

```
1. Login
2. Create New Account
3. Exit
Enter choice: 1

--- Login ---
Username: alex
Password: 123

Welcome, alex!

==========================================
   ENCRYPTED CLOUD FILE STORAGE SYSTEM
==========================================

--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice:
```

Test Case 3: FileUpload

```
Enter your choice: 1

--- Upload File ---
Filename: ProjectReport.pdf
Encryption key: 1234

File uploaded!
File ID: FILE_1764794015
Remember your ID and key!

--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice:
```

Test Case 4: File Download

(Correctkey)

```
--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice: 2

--- Download File ---
File ID: FILE_1764794015
Decryption key: 1234
Output filename: ej.pdf

File downloaded!
```

Test Case 5: File Download

(WrongKey)

```
--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice: 2

--- Download File ---
File ID: FILE_1764794015
Decryption key: y78
Wrong key!
```

Test Case 6: List Files

```
--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice: 4

--- Your Files ---
File ID              Original Name
-----------------------------------------------
FILE_1764794015      ProjectReport.pdf
```

Test Case 7: File

Deletion

```
--- MAIN MENU ---
1. Upload File
2. Download File
3. Delete File
4. List All Files
5. Exit

Enter your choice: 3

--- Delete File ---
File ID: FILE_1764794015
Enter password to confirm: 123

File deleted!
```

5.2 Memory Management Test

- Tested with files up to 100MB
- No memory leaks detected
- Proper free() after malloc()
- File handles closed properly

---

6. CONCLUSION & FUTURE WORK

6.1 Conclusion

The Encrypted Cloud File Storage System successfully implements a secure, user-friendly file management solution using C programming. The project demonstrates:

1. **Modular Design**: Clear separation of concerns with four distinct modules
2. **Security Implementation**: User authentication and file encryption
3. **File Operations**: Complete CRUD functionality for files
4. **User Isolation**: Each user has private storage space
5. **Cross-Platform**: Works on both Windows and Linux systems

Key Achievements:

- Successfully implemented XOR encryption algorithm
- Created robust metadata management system
- Developed user authentication with password hashing
- Built intuitive command-line interface
- Ensured proper memory management

Learning Outcomes:

- File handling in C (binary and text)
- Dynamic memory allocation
- String manipulation and parsing
- Modular programming with header files
- Cross-platform development
- Algorithm implementation (hashing, encryption)

6.2 Limitations

1. **Encryption Strength**: XOR cipher is basic and not suitable for production
2. **No Network Support**: Local storage only, not true cloud
3. **Single Session**: No concurrent user support
4. **File Size**: Limited by available RAM during encryption
5. **Password Recovery**: No mechanism to reset forgotten passwords

6.3 Future Enhancements Short-term Improvements:

1. Implement AES-256 encryption for better security
2. Add file compression before encryption
3. Support for file versioning
4. GUI interface using GTK or similar
5. Password strength validation

Long-term Enhancements:

1. **Network Functionality**:
   - Client-server architecture
   - Remote cloud storage
   - Multi-user concurrent access
2. **Advanced Features**:
   - File sharing between users
   - Access control lists (ACL)
   - File search and filtering
   - Backup and restore functionality
3. **Security Improvements**:
   - Two-factor authentication
   - Session management
   - Audit logs
   - Password recovery via email
4. **Performance Optimization**:
   - Streaming encryption for large files
   - Database instead of flat files
   - Caching mechanism
     - Multi-threading support

6.4 Real-World Applications

This project demonstrates concepts applicable to:

- Cloud storage services (Dropbox, Google Drive)
- Secure file transfer systems
- Backup solutions
- Document management systems
- Enterprise file servers

---

7. REFERENCES

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
2. C Standard Library Documentation. (2025). Retrieved from https://en.cppreference.com/
3. XOR Cipher Implementation. (2024). Retrieved from https://en.wikipedia.org/wiki/XOR_cipher
4. DJB2 Hash Algorithm. Retrieved from http://www.cse.yorku.ca/~oz/hash.html
5. File Handling in C. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/file-handling-c-classes/
6. Cross-Platform Programming in C. Retrieved from https://stackoverflow.com/questions/tagged/cross- platform+c
7. UPES Course Material: CSEG1032 - Programming in C (2024-25)