ShatterLock

(Established under Karnataka Act 16 of 2013)

Problem Solving with C - UE24CS151A

Feb-May, 2025

Level-2 Orange Problem Submission

Software Requirement Specification

Team Number: 5

Submitted By:

Suraj Acharya [PES2UG24CS539]

Shashidharan VS [PES2UG24CS568]

Sushanth B [PES2UG24CS543]

Sujay P Nayak [PES2UG24CS533]

Submitted To:

Dr.Gamini Joshi

Software Requirement Specification (SRS) Document

ShatterLock

1. Brief Description of Project

ShatterLock is a secure, local encryption system designed to protect user data on a computer. It uses fragmentation, multi-layered metadata encryption and distributed storage techniques to ensure encrypted files cannot be directly copied to a different system. Files once encrypted are split into packets, which are then scattered across the filesystem along with junk packets. The same process is done in reverse to decrypt and read files. The system supports multi-user access, allowing each user to maintain independently secured data spaces. The more users, the more secure the system. Hence, the security is based on both the encryption algorithm and its implementation.

2. Purpose / Goal

The goal of the project is to provide users with a strong local encryption system which is secured using an encryption algorithm and an implementation that uses fragmentation and distributed storage. Designed to be used by users with minimal technical knowledge.

3. Usefulness / Benefit

- **For Users:** The system provides users with a secure and local way to store sensitive information. It reduces the risk of unauthorised access and doesn't allow the file to be copied onto another system even if the physical device is compromised. Works for users with minimal technical knowledge.
- **For Developers:** Project showcases advanced techniques in secure storage without relying on external servers or databases. Demonstrates how fragmented encryption can be implemented at the filesystem level with pseudo-random directory selection.
- **For Learners:** Excellent learning opportunity for anybody interested in understanding encryption methods and principles, file management in C and unique ways to implement and use encryption while keeping it simple enough to be studied easily.
- **For Security Awareness:** Promotes the use of best practices to secure personalised data, like strong passwords and the need for secure local storage. Encourages users to take ownership of their security on their own computer.

4. Hardware / Software Involved

Hardware Requirements:

- Standard PC or Laptop with Linux OS
- Minimum of 2GB RAM

- Minimum of 6GB disk space
- Processor with support for basic file system operations (x64)

Software Requirements:

- Programming language: C with GCC compiler and GDB (For developers only)
- Linux Operating System (Ubuntu/Debian/Fedora) (For end users)

5.Modules Used:

- **<stdio.h>**
  - Reading and Writing into terminals and files.
- **<stdlib.h>**
  - Handles Memory Management & Pseudo-Random Number Generation.
- **<string.h>**
  - String manipulation. Aids in file naming ,key construction and operations and hashing with **<crypt.h>**
- **<stdint.h>**
  - Handles key-related operations and writing bytes into files
- **<unistd.h>**
  - Handles access to low-level OS functionality like checking if the file is present and checks the file access.
- **<sys/stat.h> [Special Module]**
  - Helps inspect file metadata to check if it's a directory.
- **<dirent.h>**
  - Handles the directory traversal & progressively scans through subdirectories as well.
- **<sys/types.h> [Special Module]**
  - Supports the other system headers with necessary types.
  - Needed alongside **<dirent.h>** and **<stat.h>**
- **<ctype.h> [Optional]**
  - Aids in character type-checking

6. Detailed Feature List

User Interaction Module:

- The user has CRUD capabilities. I.e user can create, read, update/edit and delete files.
- The user is to enter their username and password while doing any of these actions.
- During signup, the user is to enter 2-3 Security questions to provide a more complex encryption system that implements TOR-like bouncing.

Security Module:

- The files are encrypted using the username and password themselves. The locations of the files are also dependent on this. Decrypting files is impossible unless the username and password are correct. Therefore, there is no need for additional authentication.
- The files are encrypted based on a strong encryption algorithm. Then split up into many equally sized packets.
- These are then spread out into unique pseudo-random locations across the computer's files.
    a. The directories (or locations) are collected and n of them are picked in a pseudo-random method based on a key derived from the username and password. (n=number of packets)
    b. Skipping the folders which has special permissions.
- When a user requests the data,
    a. The username and password are used to generate a key, which will then be used to backtrack the location of the encrypted files.
    b. The first layer (meta-data) is decrypted to ensure the right packers are decrypted and to order the packets in the right way, making sure they are all there.
    c. The metadata is removed, and the rest of the contents are decrypted.
    d. The user can then view the files. Similarly, other CRUD operations are defined.

7. Test / Demonstration Plan

- **Unit testing:**
    - o **Encryption Module Testing**:
        - ▪ Validate that the encryption and decryption algorithms accurately transform and retrieve the original data using correct credentials.
    - o **Packetisation/fragmentation Module Testing:**
        - ▪ Confirm that encrypted files are correctly split into uniform packets without data loss or corruption and that proper norms are followed to allow for recombination.

- **File Distribution Module Testing**:
  - Verify that packets are correctly distributed across valid directories without risk of deletion and without affecting the system.
- **Key Derivation Testing**:
  - Ensure that the key generated from the username, password, and security questions is consistent and deterministic for the same inputs.

- **Integration Testing:**
  - **End-to-End Storage and Retrieval Workflow**:
    - Confirm that all individual steps work together, such as storage, retrieving the right files, reconstruction of packets and decryption.
  - **CRUD Operations Validation:**
    - Ensure the Creation, Reading, updating/editing and deletion of files are well handled.
  - **Performance:**
    - Test for maximum size encryptable without loss.
    - Increase efficiency and decrease the time taken.

- **Security Testing:**
  - **Memory:**
    - Ensure nothing more than what is needed is stored at any given moment while processing. Ex: storing passwords in plaintext in memory.
  - **Authentication:**
    - Ensure there are no edge cases where 2 users can access the same files (ex: having the same passwords.)
  - **Cryptanalysis:**
    - Try to attack the system from all angles to ensure security.
- **User Acceptance Testing:**
  - Test for ease of use for non-technical users, ensure it is intuitive to use.
  - Test for errors like entering invalid characters such as emojis, etc.

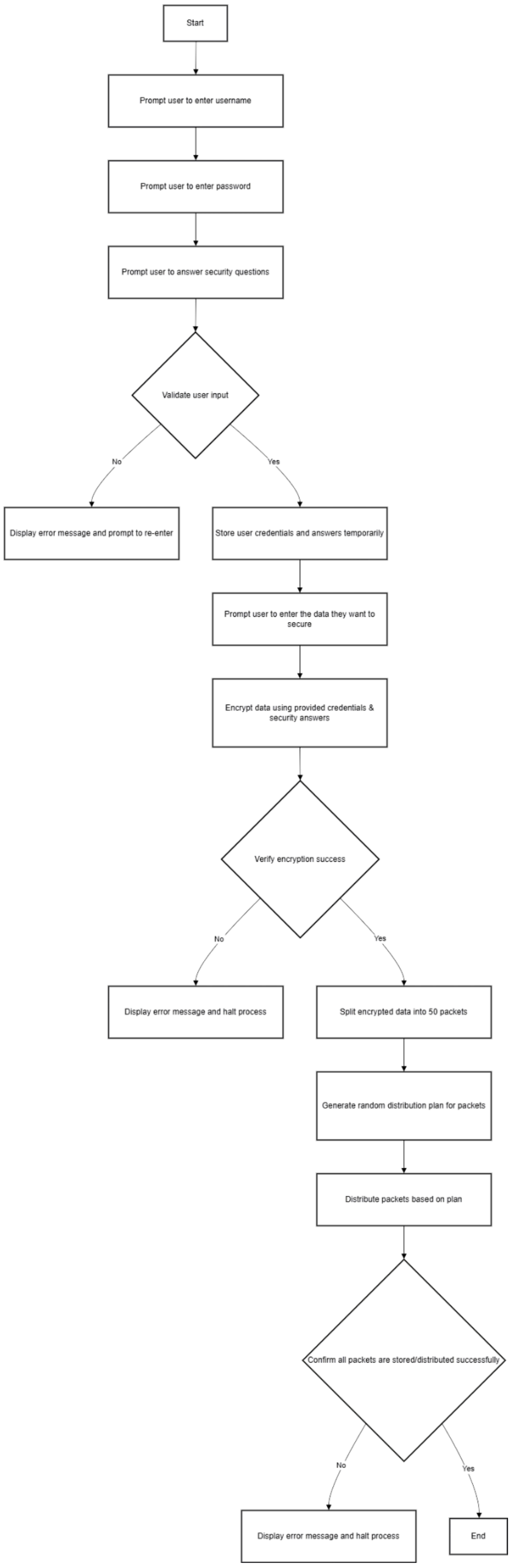
8. Expected Interaction Interface and Sample Use Cases

Interaction Interface:

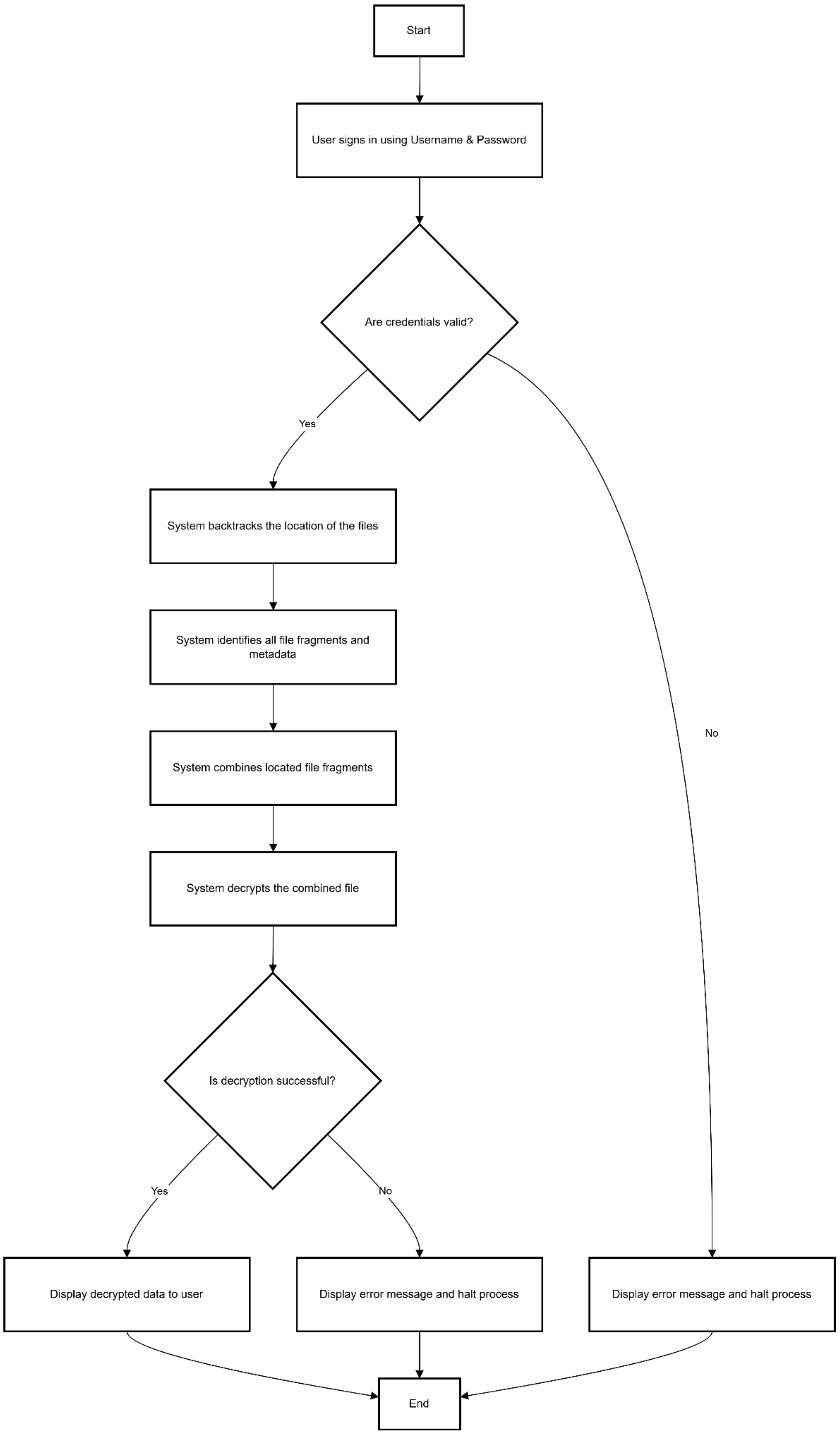
- Terminal/Command line interface for entering username, password, etc.
- Menu-driven system with CRUD options, etc.
- Hidden/masked password input.
- Warnings + confirmation before file deletion.

Sample Use Case:

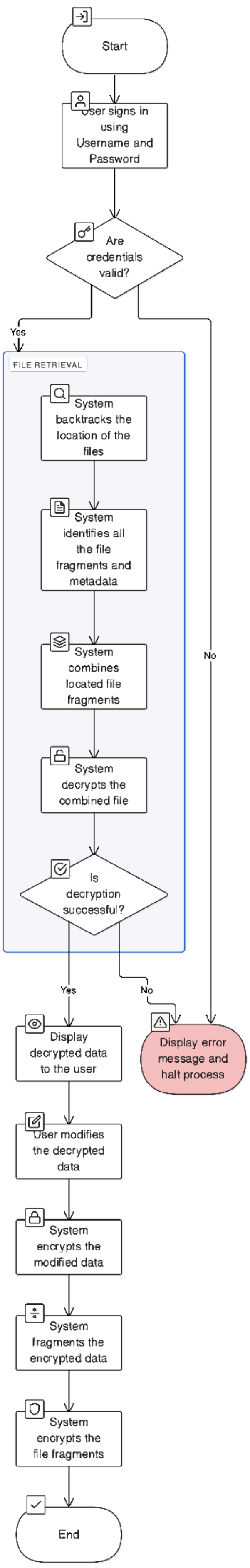
- **User registers and encrypts the file:**
    - User inputs a unique username, password, and answers security questions.
    - The security question 1 creates keys which are encrypted, fragmented and stored based on the username and password. Security question 2 is encrypted, fragmented and stored based on the username, password and key from question 1.
    - The file is then encrypted, fragmented and stored based on username, password and keys from security questions.


Start
Prompt user to enter username
Prompt user to enter password
Prompt user to answer security questions
Validate user input
No
Yes
Display error message and prompt to re-enter
Store user credentials and answers temporarily
Prompt user to enter the data they want to secure
Encrypt data using provided credentials & security answers
Verify encryption success
No
Yes
Display error message and halt process
Split encrypted data into 50 packets
Generate random distribution plan for packets
Distribute packets based on plan
Confirm all packets are stored/distributed successfully
No
Yes
Display error message and halt process
End

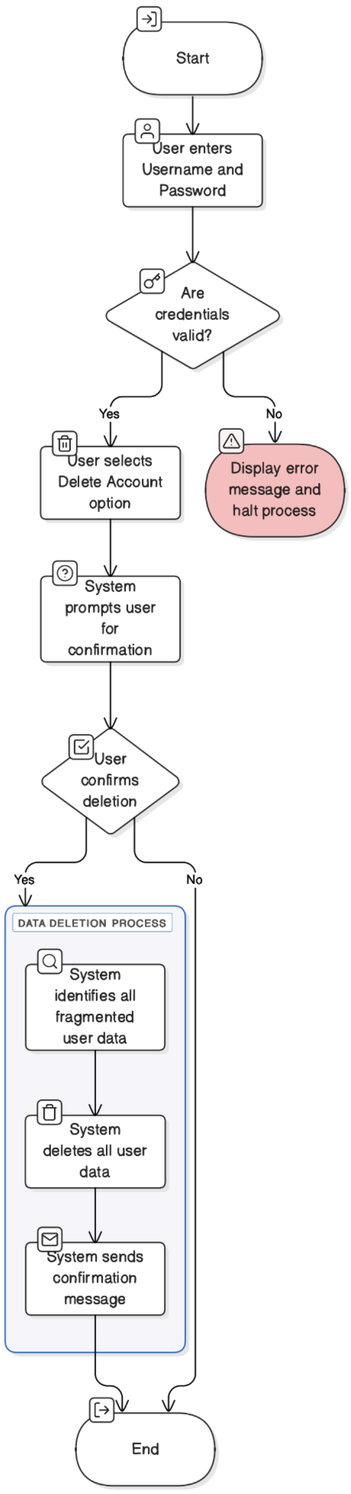
- **User reads the file:**
  - User inputs their username and password.
  - Key 1 is retrieved from username and password, pieced together and decrypted.
  - Key 2 is retrieved from username, password and key 1, pieced together and decrypted.
  - File is retrieved from username, password, key 1 and key 2, pieced together and decrypted


Start
User signs in using Username & Password
Are credentials valid?
Yes
No
System backtracks the location of the files
System identifies all file fragments and metadata
System combines located file fragments
System decrypts the combined file
Is decryption successful?
Yes
No
Display decrypted data to user
Display error message and halt process
Display error message and halt process
End

- **User edits the file:**
    - Key 1 is retrieved from username and password, pieced together and decrypted.
    - Key 2 is retrieved from username, password and key 1, pieced together and decrypted.
    - File is retrieved from username, password, key 1 and key 2, pieced together and decrypted.
    - File's packets are located and deleted.
    - Decrypted contents are edited by the user.
    - The contents are then encrypted, fragmented and stored based on username, password and keys from security questions.


Start
User signs in using Username and Password
Are credentials valid?
Yes
No
FILE RETRIEVAL
System backtracks the location of the files
System identifies all the file fragments and metadata
System combines located file fragments
System decrypts the combined file
Is decryption successful?
Yes
No
Display decrypted data to the user
Display error message and halt process
User modifies the decrypted data
System encrypts the modified data
System fragments the encrypted data
System encrypts the file fragments
End

- **User deletes the file:**
  - Key 1 is retrieved from username and password, pieced together and decrypted.
  - Key 2 is retrieved from username, password and key 1, pieced together and decrypted.
  - File’s packets are located and deleted.



Start
User enters Username and Password
Are credentials valid?
Yes
No
User selects Delete Account option
Display error message and halt process
System prompts user for confirmation
User confirms deletion
Yes
No
DATA DELETION PROCESS
System identifies all fragmented user data
System deletes all user data
System sends confirmation message
End

9. Error Handling & Edge Cases:

Errors which occur under certain circumstances/cases are common when it comes to working with files/memory management because of many reasons, which are listed below

- Incorrect username/password entered:
    - Error message to be displayed.
- Missing or Corrupted Key Packets:
    - Memory not freed correctly after encryption/decryption (memory leak).
    - Memory overflow.
    - Pointer errors when accessing fragmented keys in memory (null or dangling pointers).
- Tampering with Stored Packets:
    - Opening the fragmented key files and tampering with the data
    - Renaming or deleting the key fragment files
- Fragment Reordering:
    - Errors occur when the program is unable to reorder the fragmented keys in proper order.
- System-Dependent Path Issues:
    - New files/folders might be created, which will mess up the order of the pool of directories (SERIOUS ISSUE- needs to be addressed)
    - Permissions to the file/directory might have changed
- Large Input Data or Memory Limitations:
    - Huge amounts of data can take a lot of time and resources to encrypt and decrypt.
    - Sometimes, memory overflow might happen as well, especially when other processes are going on in the background.

*And many more.

10. Constraints/Limitations:

- **Platform dependency:**
    - Designed to run on Linux-based systems only since windows restricts access to a lot of files.
- **Local storage only:**
    - No cloud storage, no backups. Any packets deleted accidentally are not backed up, no way to recover.
- **Inefficient for a smaller number of files:**
    - Trash packets use more space than needed to ensure obscurity to the attacker.
- **User Responsibility:**
    - Username and Passwords need to be strong and kept secret.

11. Glossary of Terms

- Encryption: The process of transforming readable data into an unreadable format using an algorithm and a key.
- Metadata: Information that provides details about other data but does not contain the actual content.
- Packet: A smaller piece of data created from a larger file, used for secure storage.
- Pseudo-Random: A method of generating random numbers that appears random but is deterministically generated.
- Security Questions: Predefined questions. Answers to these are different for each user and is used for secondary verification in combination with the password

11. Conclusion.

This project aims to increase the security of local files against attackers trying to copy files onto different systems in an innovative way. While most approaches use only an encryption algorithm to ensure the security of the file, this system combines encryption with fragmentation, multi-layered encryption of metadata, filesystem obfuscation, distributed storage and TOR-like bouncing techniques. Hence, not only does it encrypt the file, but it also ensures that an attacker cannot copy it anywhere, keeping data safe even if the physical computer is compromised. It is a powerful yet user-friendly system.

12. Future Improvements.

- **Improved encryption algorithms:**
    - There are many better algorithms, and quantum resistant algorithms.
- **Advanced Redundancy:**
    - Implement error-correcting codes so that decryption is possible even if a few packets are lost or corrupted.
    - Backup data on the cloud and maintain sync.
- **Multi-User Environment Support:**
    - Extend support for secure key sharing between multiple users on the same machine without overlapping or overwriting each other's fragments.
    - Implement collaboration on shared files between two users.
- **Tampering Detection:**
    - Integrate tamper-detection mechanisms like fragment signing or timestamp validation to detect and block forged fragments.
- **Portable Mode:**
    - Allow for some files to be stored on a USB/removable device while keeping files such as both keys from the security questions on the device. That way

the files can be carried around but can only be opened by systems containing the two keys.

- **GUI Improvements**
- **Hardware Binding :**
  - Bind packet recovery to certain machine identifiers (like UUIDs or hardware IDs) to prevent key retrieval if fragments are copied to a different machine.
- **Self-Destruct Mode:**
  - Add an optional feature where, after multiple failed decryption attempts, packets auto-delete themselves to prevent brute-force attacks.