CryptBox - Software Requirements Specification

November 27, 2024

Contents

1 Introduction and Team Details

1.1 Team Members

Name	Roll No.	Email
R Sreenivasa Raju	IMT2023122	Sreenivasa.Raju@iiitb.ac.in
K Shashidhar Sai	IMT2023567	Katakam.Shashidhar@iiitb.ac.in
N Sai Sandeep	IMT2023574	SaiSandeep.Nori@iiitb.ac.in
Hardhik Dhavala	IMT2023579	Hardhik.Dhavala@iiitb.ac.in
Abhinav Kishan	IMT2023580	Abhinav.Kishan@iiitb.ac.in
CH Lithin Sai Kumar	IMT2023598	Lithin.SaiKumar@iiitb.ac.in

1.2 Project Overview

CryptBox is a command-line interface (CLI) based file management system designed to provide users with a secure, locally-managed file storage solution. The primary purpose of the application is to offer enhanced data protection and organizational capabilities through comprehensive encryption and file management features.
Key High-Level Functionalities:

1. Multi-user file system with isolated storage
2. Granular file and folder encryption/decryption
3. Dynamic file and folder labeling
4. Advanced search and filtering capabilities
5. Secure, native implementation ensuring data privacy

1.3 Scope

1.3.1 Included Features

- Support for multiple user account creation with isolated storage for each user.
- Granular encryption and decryption of files and folders, ensuring enhanced security.
- Search functionality based on user-defined labels and filters, including encryption status and file extensions.
- File and folder sharing between users with proper access control.

1.3.2 Excluded Features

- Network-based file sharing
- Cloud storage integration

- Advanced access control beyond user-level permissions
- Graphical User Interface (GUI); the application remains CLI-based.

2 Objectives

The primary objectives of CryptBox are to:

1. Develop a secure, custom file system with native encryption support
2. Provide users with granular control over their file storage
3. Implement a robust command-line interface for file management
4. Ensure data protection through strong encryption mechanisms
5. Create an intuitive and user-friendly file management experience

3 System Overview

3.1 Technical Specifications

Backend Java

Native Implementation C++

Encryption AES-256 GCM

Interface Command-Line Interface (CLI)

Platform Local file system

3.2 Input/Output Requirements

3.2.1 Inputs

- File and folder names
- Encryption/decryption keys
- File content
- CLI commands

3.2.2 Outputs

- File system operations results
- Encrypted/decrypted files
- File and folder listings
- Operation status messages

4 Functional Requirements

4.1 Key Features

4.1.1 User Management

- User registration
- User-specific root directories
- Secure password management

4.1.2 File Management

- Create files and folders
- Copy and move files/folders
- Delete files and folders
- Rename files and folders
- Add and remove labels

4.1.3 Encryption Capabilities

- File-level encryption
- Folder-level encryption
- Recursive encryption for nested folders
- Secure key management

4.1.4 Search and Filtering

- Search files and folders by label
- Filter encrypted files from all the files in the current directory
- File type filtering (Extensions)

4.2 Use Cases

4.2.1 Use Case 1: User File Creation

Registered User

1. User logs into the system
2. User navigates to desired directory
3. User creates a new file
4. System confirms file creation

4.2.2 Use Case 2: File Encryption

Registered User

1. User selects a file or folder
2. User provides encryption key
3. System encrypts the selected item
4. System confirms encryption status

5 System Setup Requirements

5.1 Precondition

Before running the CryptBox application, the following setup steps must be completed:

- Create a folder named `System` at a preferred location on the system.
- This folder will act as the root directory for the CryptBox storage system.

5.2 Configuration

The absolute path of the `System` folder must be specified in the `StorageNode` class as the value of the `truePath` attribute.

- **File:** `StorageNode.java`
- **Attribute:** `truePath`
- **Example Configuration:**

```
public static final String truePath = "C:\\CryptBox\\System";
        // Update with the correct path
```

Ensure that the `System` folder exists at the specified location before running the application.

6 Non-functional Requirements

6.1 Performance Requirements

- Quick file system operations (¡ 1 second for small files)
- Minimal performance overhead during encryption/decryption

6.2 Usability

- Clear and concise CLI commands
- Intuitive command syntax
- Concise error messages
- Can get used to the CLI commands quickly

6.3 Security Requirements

- AES-256 GCM encryption
- Secure key generation and management
- Protection against unauthorized access
- Only uses the hashed form of passwords

7 Development Setup

7.1 Requirements

- Java Development Kit (JDK) 8 or higher
- C++ Compiler (GCC 9+ or Clang with the support for c++17 or later)
- Operating system: macOS, Windows
- Git

7.2 Setup Instructions

```
# Clone the repository
git clone https://github.com/ChLithin/CryptBox.git

# Navigate to project directory
cd CryptBox
```

```
# Run the following command
build.bat (On Windows)
    or
build.sh (On macOS)
```

8 Workflow

8.1 Backend Logic

- Implement file system operations in Java
- Use native methods written in CPP for better performance
- Manage user-specific storage spaces
- Handle encryption and decryption processes (Written in Java)

8.2 CLI Interaction

- Implement command parsing
- Process user commands
- Provide feedback and handle errors
- Manage user sessions

9 Important Files & Folders

CustomFile.java Core file management class

CustomFolder.java Folder management and organization

User.java User account and authentication

StorageNode.java Base class for file system elements

CLI.java Command-line interface handler

MainCLI.java User registration and management

CustomFile.cpp Implementation of file operation functions using JNI

CustomFolder.cpp Implementation of folder operations using JNI

10 Testing & Logging

10.1 Testing Strategy

- **Unit Testing:** Validate core methods such as `touch`, `encrypt`, and `ls` for correctness.
- **Integration Testing:** Ensure seamless operation between Java and C++ modules via JNI and verify complete workflows.
- **Boundary Testing:** Test edge cases like large files, deeply nested folders, and invalid command inputs.
- **Security Testing:** Confirm encryption key confidentiality and simulate unauthorized access attempts to evaluate system resilience.

10.2 Logging

- **User Actions:** Log activities such as login, logout, file creation, encryption, and sharing.
- **Error Logs:** Record failed commands, encryption errors, and invalid inputs, including detailed stack traces.
- **Audit Trails:** Maintain chronological logs of critical operations for system monitoring and debugging.

11 Conclusion

CryptBox offers a secure and user-centric file management solution tailored for environments where multiple users share a single system. By providing each user with an isolated, custom-designed file system, CryptBox ensures privacy and autonomy in managing files and folders. Key features such as granular encryption, dynamic labeling, and search capabilities empower users to securely organize and access their data.

The application's CLI-based design is intuitive yet powerful, allowing users to perform complex operations with ease. Backed by native C++ integration for performance and AES-256 encryption for security, CryptBox stands out as a reliable tool for local file management. Its focus on individual user workspaces ensures that multiple users can collaborate on the same system without compromising their personal data or workflow.