ECE 422 – Group Project: Secure File System

Group Name: The ByteKnights

Name	SID	CCID
Parth Dadhania	1722612	pdadhani
Het Bharatkumar Patel	1742431	hetbhara

Abstract

This document presents the initial design and development approach of the **Secure File System (SFS)** project. The goal of SFS is to provide an environment where multiple internal users can securely create, store, and share text files on an untrusted file server. Through robust authentication, encrypted file names and contents, and Unix-like group and permission controls, the system ensures both confidentiality and controlled access. Additionally, integrity checks, facilitated by cryptographic mechanisms (e.g., HMAC), detect unauthorized tampering, notifying file owners upon login. The project incorporates a command-line interface (CLI) for core file operations (e.g., pwd, ls, cd, touch, etc.) and a back-end that handles encryption, permissions, and metadata management. This deliverable details the planned architecture, chosen technologies, and user interaction scenarios, reflecting the foundational progress toward building a reliable and secure file system.

Introduction

Data storage on untrusted servers is increasingly common, and ensuring the security of stored data has become a critical challenge. Traditional file systems primarily rely on operating system-level permissions, which can be insufficient in hostile or untrusted environments. The **Secure File System (SFS)** is designed to mitigate these risks by providing encryption of filenames and file contents, robust access controls, and integrity checks to detect any unauthorized modifications. Drawing inspiration from Unix file and directory permissions, SFS extends these concepts with cryptographic safeguards, allowing multiple internal users to collaborate securely while preventing external users from accessing or tampering with the data.

A key requirement of this system is **multi-user group management**, where user roles and group memberships govern reading and writing privileges. The SFS architecture incorporates password-based user authentication, ensuring only authorized personnel gain entry to the system. In addition, the command-line interface (CLI) offers familiarity with common file operations (e.g., pwd, ls, cd, touch, cat, etc.), making the system approachable for end users.

In summary, the SFS project addresses three core security pillars:

1. **Confidentiality:** Encryption of filenames, directory structures, and file content.
2. **Integrity:** Cryptographic checks (e.g., HMAC) to detect tampering and alert owners immediately upon login.
3. **Controlled Access:** Unix-like permissions enforced via owner/group/other modes, supplemented with user authentication and group membership.

Through this project, we aim to provide a scalable and demonstrably secure approach to storing sensitive data on an untrusted file server, showcasing how design choices in encryption, authentication, and permission handling can be combined to create a robust solution.

Planned Design

1. Technologies, Methodologies, and Tools

1.1 Programming Languages & Frameworks

- **Python**:
    - Chosen for its simplicity and extensive library support.
    - Rich ecosystem of cryptographic libraries (e.g., cryptography), enabling robust encryption and integrity checks.
    - Facilitates rapid development and easy creation of a command-line interface.
- **Cryptography Libraries** (e.g., cryptography package in Python):
    - Offers high-level APIs for AES encryption, HMAC, and key management.
    - Actively maintained and well-documented, ensuring reliability for production-grade security.
- **Password Hashing** (bcrypt or hashlib with PBKDF2):
    - Provides salted, one-way hashing to securely store user passwords, preventing plain-text credential leaks.
- **Data Storage** (Encrypted JSON/SQLite):
    - JSON or SQLite can store user/group metadata, file permissions, and encryption keys.
    - Both are light in overhead and easy to integrate with Python.
    - Secured via encryption at rest to prevent unauthorized data exposure.

1.2 Methodologies

- **Agile** (Incremental Approach):
    - The project is divided into short sprints, each targeting a specific set of features (e.g., authentication, encryption, integrity checks).

    - Regular sprint reviews and demonstrations ensure ongoing alignment with requirements and provide frequent feedback loops.
    - Promotes iterative improvements, making it easier to pivot if technical or design challenges arise.
- **Test-Driven & Documentation-Driven Development** (where feasible):
    - Core functionality is incrementally tested, ensuring early detection of bugs.
    - Design diagrams (high-level architecture, class, sequence) are iteratively refined to reflect the evolving system.

1.3 Tools

- **Version Control**:
    - Git + GitHub for source code management, enabling collaborative development and code review.
- **Continuous Integration**:
    - GitHub Actions to automate tests on each commit or pull request.
    - Ensures that newly introduced changes do not break existing features.

By combining these technologies, methodologies, and tools, we aim to build a robust, maintainable, and secure file system that complies with the project's requirements around confidentiality, integrity, and controlled access.

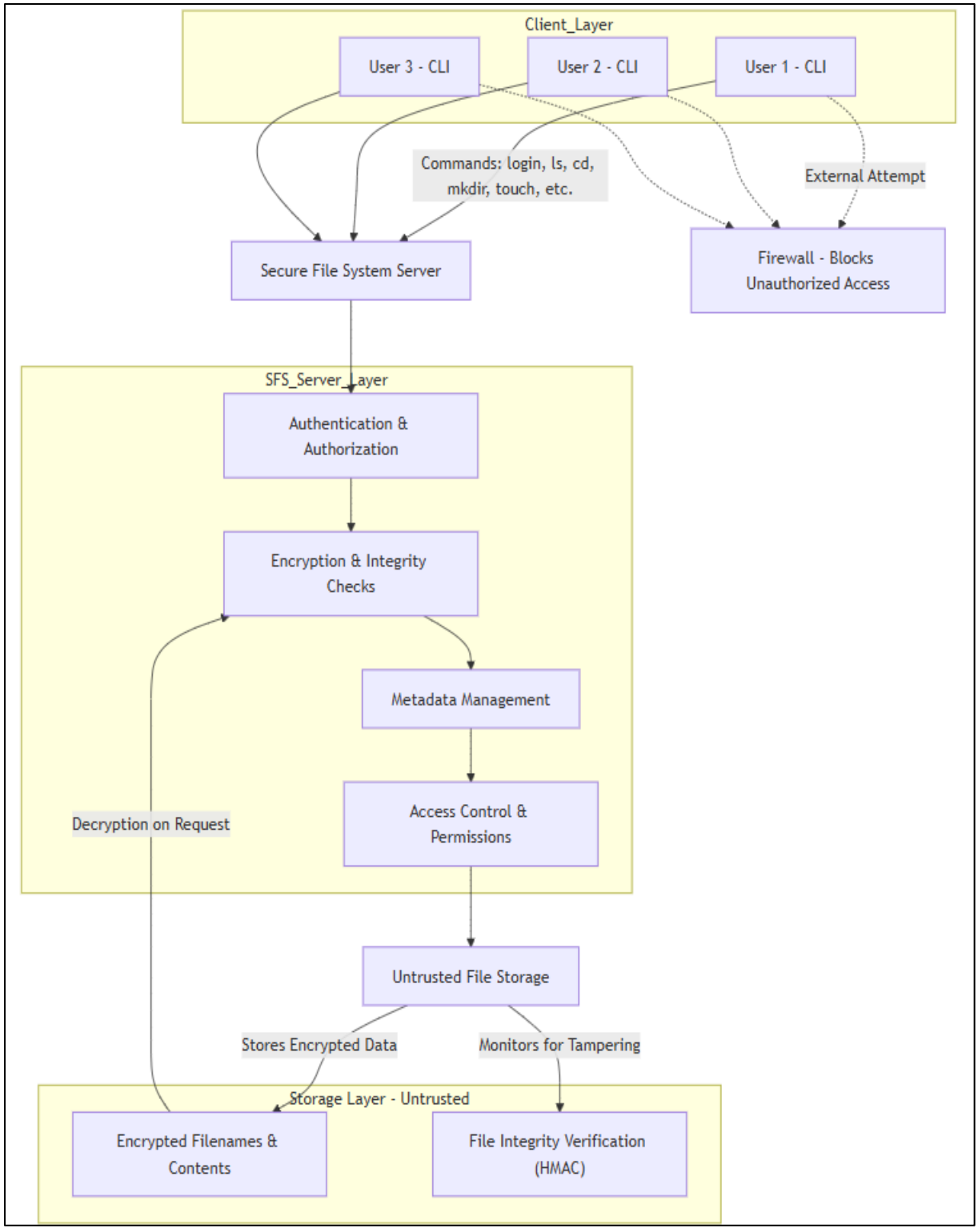

Figure 1. High-Level Architecture of the Secure File System (SFS) – Showcasing user interaction, authentication, encryption, access control, and secure storage.

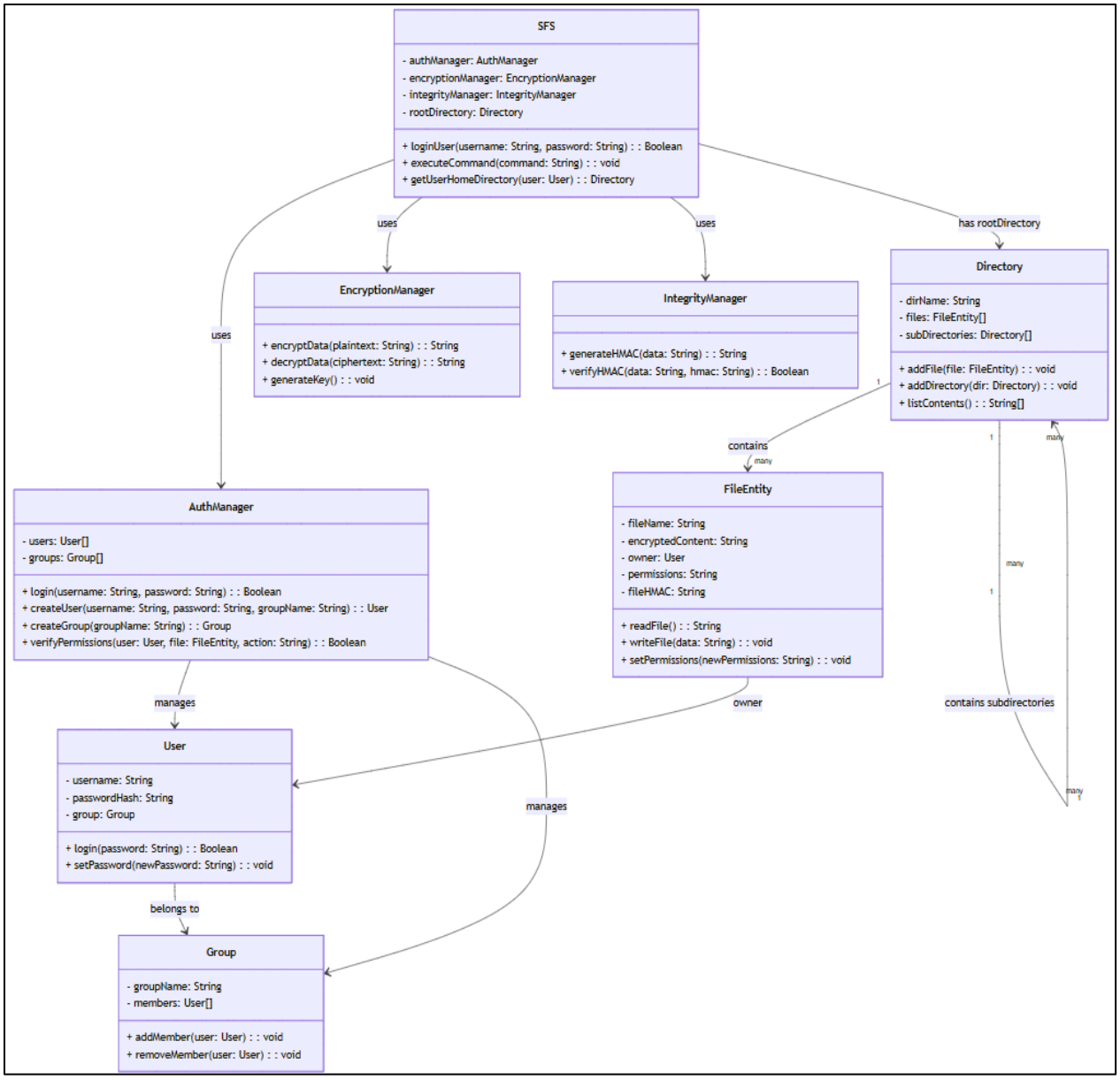

Figure 2. UML Class Diagram of the Secure File System (SFS) – Showcasing key classes, attributes, methods, and relationships, including user authentication, encryption, access control, and hierarchical file management.

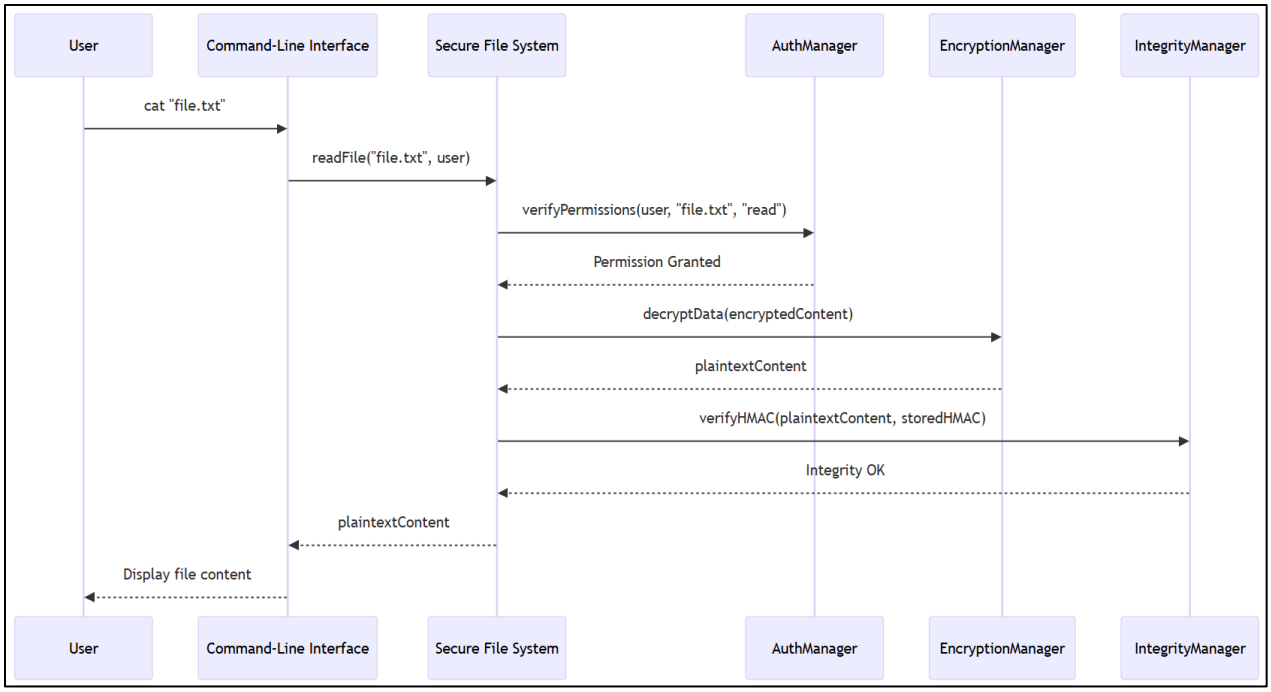

Figure 3. Sequence Diagram for File Read Operation in Secure File System (SFS) – Illustrates step-by-step interactions for authentication, permission verification, decryption, and integrity checks before displaying file content.

User Stories

1. User Account Creation

User Story: As a new user, I want to create an account so that I can securely store and manage my files.

Acceptance Criteria

1. The system prompts for a unique username and secure password (hashed, not stored in plaintext).
2. If the username is already taken, the system notifies me and requests a different one.
3. Upon successful creation, I should be able to log in and access a personal home directory.

2. Group Management

User Story: As a team lead, I want to create a group and add/remove members so that we can collaborate on files under shared permissions.

Acceptance Criteria

1. Only authorized users (e.g., an admin or group creator) can create a group.
2. The system prevents duplicate group names and returns an error if a group name already exists.
3. Group members can read/write files if granted appropriate permissions, and non-members cannot.

3. File Operations

User Story: As an authenticated user, I want to create, read, write, and rename files so that I can manage my data within the secure file system.

Acceptance Criteria

1. The system encrypts filenames and contents on disk automatically.
2. When I create a file, I become its owner and can set initial permissions (owner/group/other).
3. If I rename a file, the system updates the encrypted filename accordingly, preserving its contents and metadata.
4. Unauthorized users cannot read or modify my files based on permission checks.

4. File Integrity & Tamper Detection

User Story: As a file owner, I want to be notified if my files are tampered with so that I can take action and maintain data integrity.

Acceptance Criteria

1. The system generates an HMAC for each file and verifies it upon login or file access.
2. If an external user modifies a file's contents outside of SFS, the system detects the mismatch and alerts the owner immediately.
3. The alert includes which file(s) were tampered with, and the file remains in a corrupted state until the owner takes manual corrective actions (e.g., restoring from a backup).

5. Encrypted Visibility for External Users

User Story: As an external (untrusted) user, I want to see only encrypted data so that I cannot read or meaningfully modify SFS files.

Acceptance Criteria

1. Any file or directory names appear garbled or encrypted to me if I am not authenticated as an internal user.
2. The file contents remain unreadable when accessed via conventional OS tools.
3. If I modify an encrypted file directly, the system detects the tampering on the next login by the file's owner.