Secure Enclave for Musical Artefacts – Final Submission Report

1. Overview

The Secure Enclave application is a Python 3 command-line tool designed for the secure storage of encrypted artefacts (e.g., musical scores, lyric sheets, and MP3 files). It provides complete CRUD (Create, Read, Update, Delete) functionality alongside role-based access control (RBAC). The application employs robust security protocols—including password hashing, file encryption, and checksum verification—to ensure the principles of Confidentiality, Integrity, and Availability (CIA) (Chitadze, 2023).

The final implementation was based on the Unit 3 design document, which included UML use case, class, and sequence diagrams to represent system requirements and interactions. The design emphasised modularity, object-oriented methodologies, and comprehensive security practices (Flageol, 2023).

2. Comparison: Unit 3 Design versus Unit 6 Implementation

a. Architectural Design and Modularity

Unit 3 Design:
The design document presented UML diagrams (use case, class, sequence) to outline the operational framework of a secure digital application. It highlighted:

- **Modularity and Extensibility:** By dividing storage, encryption, and RBAC into separate components.
- **Design Patterns:** For example, the Strategy Pattern facilitated interchangeable encryption and storage mechanisms, while constructor injection was used for decoupling dependencies.

Unit 6 Implementation:
The final code is organised into concrete classes:

- **dbadmin:** Manages the SQLite connection using a singleton-like approach.
- **encryptionadmin:** Handles key management and file encryption/decryption using Fernet (which utilises AES-256 encryption).
- **uservault and artefactvault:** Act as repositories to encapsulate all database operations.
- **arterfactadmin:** Oversees file storage and retrieval.

This implementation reflects the Unit 3 design while streamlining certain patterns—for example, employing concrete classes rather than interfaces—to reduce complexity while still preserving modularity and security (Alami et al., 2023).

b. Security Measures

Unit 3 Design:
The initial design specified the implementation of robust encryption (AES-256), checksum validation (SHA-256), and strict RBAC to reduce the risk of unauthorised access. It also advised incorporating audit logging to monitor critical operations.

Unit 6 Implementation:
Key security improvements include:

- **Password Hashing with bcrypt:**
  Passwords are hashed using bcrypt prior to storage, ensuring that plaintext passwords are never retained.
- **File Encryption:**
  Files are encrypted using the Fernet implementation from the cryptography library, thus ensuring confidentiality.
- **Checksum Calculation:**
  Every file has an automatically computed SHA-256 checksum to verify its integrity.
- **Exception Handling:**
  Essential operations (database, file I/O, encryption/decryption) are encapsulated within try/except blocks to prevent leakage of sensitive error information.
- **Role-Based Access Control:**
  The CLI requires authentication for every artefact operation, ensuring that only administrators or the artefact owner can modify or view data.

While the original design recommended extensive audit logging, the final implementation outputs status messages and errors to standard output. This approach can be extended with a dedicated audit logger in future updates (Găitan & Zagan, 2021).

c. Data Structures and Data Flows

Unit 3 Design:
The design specified the use of a SQL database to store user data and artefact metadata—including file paths, checksums, and timestamps—with detailed data flows illustrated by sequence diagrams.

Unit 6 Implementation:

- **Database Schema:**
  The SQLite database includes two tables: users (containing username, hashed password, and role) and artefacts (containing owner_id, file_name, encrypted file_path, checksum, created_at, and updated_at).
- **Data Flow:**
  For artefact operations (e.g., upload), the flow is as follows:
    1. User authentication.
    2. File encryption.
    3. Checksum calculation.
    4. Metadata storage in the database.

This flow adheres to the sequence diagram from Unit 3 and ensures both data integrity and secure access.

d. API and Command Interface

Unit 3 Design:
The design document outlined a RESTful approach for managing artefacts; however, for this assignment a CLI was required. It also emphasised distinct roles for administrators and standard users.

Unit 6 Implementation:
The final CLI application supports the following commands:

- **register:** Create a new user.
- **login:** Authenticate a user.
- **add:** Add a new artefact.
- **view:** View an artefact.
- **update:** Update an artefact.
- **delete:** Delete an artefact.
- **list:** List artefacts.

User authentication is prompted before performing artefact operations to enforce RBAC, which meets the CRUD requirements and aligns with the original design's objectives (Kabier et al., 2023).

3. Testing, Automated Tools, and Evidence

a. Unit Testing

Unit tests verify core functions, such as user authentication and checksum calculation. These tests were developed using Python's unittest framework and can be executed with:

```
python -m unittest discover
```

This confirms that the application implements its intended functionality.

b. Static Analysis

- **PyLint:**
  PyLint was run on the code. The detailed report is saved as pylint_report.txt.

```
pylint app.py > pylint_report.txt
```

- **Flake8:**
  Flake8 was used to ensure adherence to coding standards:

```
flake8 app.py > flake8_report.txt
```

c. Security Scanning

- **Bandit:**
  The code was scanned with Bandit:

bandit -r . > bandit_report.txt

The scan reported no critical security issues, indicating that secure coding practices were successfully implemented.

4. Environment Setup and Running Instructions

Virtual Environment and Dependencies

1. **Create and Activate a Virtual Environment:**

python3 -m venv venv

source venv/bin/activate

2. **Install Dependencies:**

pip install cryptography bcrypt

Alternatively, use a requirements.txt file containing:

cryptography

bcrypt

and install with:

pip install -r requirements.txt

Running the Application

Use the following commands for different operations:

- **Register a User:**

python app.py register --username testuser --password testpass --role user

- **Login:**

python app.py login --username admin --password adminpass

- **Add an Artefact:**

python app.py add --file /path/to/file --name "MySong"

- **View, Update, Delete, and List Artefacts:**
  Refer to the command documentation below.

5. Command Documentation

register

- **Purpose:** Register a new user with a specified role (admin or user) using bcrypt for password hashing.
- **Usage:**

python app.py register --username <username> --password <password> --role <admin|user>

- **Example:**

python app.py register --username testuser --password testpass --role user

login

- **Purpose:** Authenticate an existing user.
- **Usage:**

python app.py login --username <username> --password <password>

- **Example:**

python app.py login --username admin --password adminpass

add

- **Purpose:** Add a new artefact by encrypting the file, computing its SHA-256 checksum, and storing metadata.
- **Usage:**

python app.py add --file <path_to_file> --name <artefact_name>

- **Example:**

python app.py add --file /home/user/music/song.mp3 --name "MySong"

view

- **Purpose:** View artefact details and preview the decrypted file content.
- **Usage:**

python app.py view --id <artefact_id>

- **Example:**

python app.py view --id 1

update

- **Purpose:** Update an existing artefact with a new encrypted file.
- **Usage:**

python app.py update --id <artefact_id> --file <path_to_new_file>

- **Example:**

python app.py update --id 1 --file /home/user/music/new_song.mp3

delete

- **Purpose:** Delete an artefact and remove its encrypted file.
- **Usage:**

python app.py delete --id <artefact_id>

- **Example:**

python app.py delete --id 1

list

- **Purpose:** List all artefacts accessible to the authenticated user.
- **Usage:**

python app.py list

- **Example:**

python app.py list

6. Justification of Deviations from the Unit 3 Design

Various modifications were made to the original Unit 3 design to improve security and maintainability:

- **Password Hashing:**
  The final implementation utilises bcrypt for password hashing rather than storing plaintext passwords, thereby significantly enhancing security.
- **Object-Oriented Refactoring:**
  The design was refactored into concrete classes instead of using interfaces and constructor injection. This reduction in complexity preserves modularity while making the code easier to maintain.
- **Simplified Audit Logging:**
  Although the initial design proposed comprehensive audit logging, the final implementation outputs status messages to standard output. This decision was made to streamline the CLI application, with the possibility of extending audit logging in future updates.
- **CLI Interface:**
  While the Unit 3 design anticipated a more RESTful API approach, the final product is a CLI application as required. This deviation was necessary to comply with the assignment brief and to avoid using external frameworks such as Django or Flask.

7. Conclusion

This final submission presents a secure, modular, and robust CLI application for managing encrypted musical artefacts. The application adheres to the original design brief by implementing:

- **Secure Storage:** File encryption using AES-256 via Fernet and SHA-256 checksums.
- **Robust Access Control:** Role-based operations enforced through user authentication.
- **Modular, Object-Oriented Design:** Clear separation of concerns through classes such as dbadmin, encryptionadmin, uservault, and artefactvault.

- **Automated Testing and Security Analysis:** Evidence of unit testing, static analysis (PyLint, Flake8), and security scanning (Bandit) confirms high code quality and compliance with secure coding practices.

References

Alami, A., Zahedi, M. & Krancher, O., 2023. Antecedents of psychological safety in agile software development teams. *Information and Software Technology*. Available at: https://www.sciencedirect.com [Accessed 15 February 2025].

Chitadze, N., 2023. Basic principles of information and cyber security. In: *Analyzing New Forms of Social Disorders in Modern Virtual Environments*, pp. 193–223. IGI Global.

Flageol, W., 2023. Improving Object-Oriented Programming by Integrating Language Features to Support Immutability. Available at: https://www.concordia.ca [Accessed 15 February 2025].

Găitan, V.G. & Zagan, I., 2021. Experimental implementation and performance evaluation of an IoT access gateway for the Modbus extension. *Sensors*. Available at: https://www.mdpi.com [Accessed 15 February 2025].

Kabier, M.K., Yassin, A.A., Abduljabbar, Z.A. & Lu, S., 2023. Role Based Access Control Using Biometrics in Educational Systems. *Basrah Researches in Sciences*. Available at: https://www.iasj.net [Accessed 15 February 2025].