University of Regina | Go far, together.

ENSE 375 – Software Testing and Validation

VaultGuard

Brydon Herauf (200454546)

Ansar Ahmed (200470692)

Gursharan Singh Rehal (200480626)

Table of Contents

List of Figures

List of Tables

1 Introduction

This section will be built out as the report grows.

2 Design Problem

2.1 Problem Definition

Cyberattacks and credential-based breaches are becoming increasingly common in our digital world. Managing passwords securely is more challenging than ever for individuals and organizations. As a result, people often resort to weak or reused passwords. According to [1], 65% of individuals recycle passwords across different sites. This practice increases the risk of being affected by a breach. Our project addresses these issues by developing a secure and user-friendly password management software. It will allow users to generate, store, organize, and retrieve keys conveniently and safely.

2.2 Design Requirements

2.2.1 Functions

VaultGuard must contain the following functionality:

- Encrypts and stores user keys securely
- Generates strong passwords
- Registers, authenticates, and logs out users
- Stores keys on a per-user basis
- Allows users to view, copy, add, edit, and delete keys
- Organizes keys into user-defined categories

If time permits, the application may also do the following:

- Implements multi-factor authentication
- Tracks and displays logs of user activity

2.2.2 Objectives

VaultGuard must possess the following qualities:

- Secure - The system must use encryption for data in transit (if using an API) and at rest. Unauthorized access to keys must be forbidden.
- User-Friendly - VaultGuard must be intuitive and follow people-centered design principles.

- Scalable - Code shall be written in a way that supports multiple users and growing password lists without requiring major refactoring.
- Maintainable - Code shall be well- structured, and easy to add features like multi-factor authentication.
- Testable - The system shall be built using test-driven development processes. Code must support intuitive and comprehensive testing.
- Reliable - The system must function correctly at all times to the best of our abilities, and gracefully handle errors.
- Efficient - The system shall be light-weight, running quickly with limited resources.

2.2.3 Constraints

VaultGuard must conform to a number of constraints, including:

Economic Factors:

- Must be built using custom software and free libraries so the development cost is $0.
- The system must be usable on systems without a dedicated GPU so users do not have to purchase powerful hardware.

Regulatory Compliance:

- Data at rest shall be encrypted using AES-256.
- Passwords and cryptographic keys shall never be stored in plaintext.
- Access to decrypted credentials must be restricted to authenticated users only.

Reliability:

- System shall have a measured uptime of over 99%.
- Storage of unvalidated or improperly formatted passwords must be prevented.
- Credentials must be decrypted accurately in over 99% of test cases.

Sustainability and Environmental Factors:

- The application must avoid GPU requirements to support energy-efficient use on general-purpose hardware.

Ethics and Societal Impacts:

- User data must not be transmitted to external servers.
- The final product must be open-source and freely available on GitHub.

3 Solution

Our team explored multiple design approaches before settling on our final solution. While the first two approaches provided some of the desired functionality, they had serious flaws in terms of user experience, testability, and compliance with constraints. We improved upon this in solution three, and will be moving forward with development.

3.1 Solution 1

We began our brainstorming with the most basic possible implementation. In this version:

- A JSON file would contain a list of plaintext key-value pairs.
- A single script would provide a command-line interface for adding, deleting, copying, and viewing keys.

This solution follows a monolithic architecture, with no module separation of the user interface, storage logic, and input handling. While this approach would be easy to implement, it lacks critical features, fails on nearly every non-functional requirement, and violates a number of constraints. It is not secure as there is no encryption or authentication. It cannot support multiple users, and the use of a single file makes the application not scalable at all. Most importantly, it is extremely difficult to test:

- Path testing becomes impractical due to the large number of branches and tightly coupled logic.
- Integration testing becomes the only option as the lack of modularity prevents individual unit testing.

Despite its limitations, this version helped us establish a mental model for the application's core functionality, and acted as a good starting point for further discussion.

3.2 Solution 2

After ruling out the monolithic approach, we explored a more modular architecture. This improved solution contains:

- Support for multiple users through registration and login mechanisms.

- Enhanced security by encrypting each key individually before storage.
- Separate modules for managing users, encryption/decryption, key storage, and command-line interaction.

This solution meets most of our functional requirements, and significantly improves on our objectives. The addition of authentication and encryption makes it far more secure. Introducing basic modularity is a major step forward for scalability, maintainability, and testability. The most important improvement is the decoupling of components, which allows for unit testing. However, this solution still has many limitations:

- The command-line interface is not user-friendly and remains tightly coupled with encryption and storage logic.
- Coupling between modules makes path testing difficult and unit testing impractical in some areas.
- It is challenging to fully test all possible command sequences and usage scenarios.

Our second solution brought us much closer to satisfying the requirements, objectives, and constraints. However, it still falls short of our goals for robust testing and clean modular design. As a result, we proceeded with the design of a third and final solution.

3.3 Final Solution

This is the final solution. Explain why it is better than other solutions (focus more on testing). You may use a table for comparison purposes. After providing the reason for selecting this solution, detail it below.

3.3.1 Components

What components you used in the solution? What is the main purpose of using individual component? What testing method did you employ for each component? Provide a block diagram (with a numbered caption, such as Fig. 1) representing the connectivity and interaction between all the components.

3.3.2 Environmental, Societal, Safety, and Economic Considerations

Explain how your engineering design took into account environmental, societal, economic and other constraints into consideration. It may include how your design has positive contributions to the environment and society? What type of economic decisions you made? How did you make sure that the design is reliable and safe to use?

3.3.3 Limitations

Every product has some limitations, and so is the case with your design product. Highlight some of the limitations of your solution here.

4 Team Work

4.1 Meeting 1

Date: Thursday, June 5, 2025.

Agenda: Brainstorming of implementation strategies

Team Member	Previous Task	Completion State	Next Task
Brydon	N/A	N/A	N/A
Ansar	N/A	N/A	N/A
Gursharan	N/A	N/A	N/A

4.2 Meeting 2

Time: Month Date, Year, hour: minutes am/pm to hour: minutes am/pm

Agenda: Review of Individual Progress

Team Member	Previous Task	Completion State	Next Task
Team member 1	Task 1	80%	Task 1, Task 5
Team member 2	Task 2	50%	Task 2
Team member 3	Task 3	100%	Task 6

4.3 Meeting 3

Provide a similar description here.

4.4 Meeting 4

Provide a similar description here.

5 Project Management

Gantt chart has been created under our GitHub repo but development tasks are not yet mapped out.

6 Conclusion and Future Work

- A summary of what you achieved. Mention all the design functions and objectives that you achieved while satisfying testing requirements?
- While keeping the limitations of your solution, provide recommendations for future design improvements.

7 References

[1] Enzoic, “8 Shocking Stats on Password Reuse,” *Enzoic Blog*, May 11, 2022. [Online]. Available: https://www.enzoic.com/blog/8-stats-on-password-reuse/

8 Appendix

If you want to provide an additional information, use this appendix.