Software Requirement Specification (SRS)

For

Development of PassGuard-CLI

Version 1

Submitted to:

Md. Mahmodul Hasan
Lecturer, Department of Software Engineering
University of Frontier Technology, Bangladesh.
mahmodul0001@bdu.ac.bd

Prepared By:

Irshad Hossain
ID: **2303030**
Student, Department of Software Engineering
University of Frontier Technology, Bangladesh.
irshadrisad@gmail.com

PassGuard-CLI SRS Approval Letter

I, the candidate, understand that this SRS contains all the software requirement details that have been clearly written for explaining how PassGuard-CLI will work in the real world. I also understand that after submitting this SRS, I will not be able to add more requirements to PassGuard-CLI. I am giving my request and approval to my lecturer, Md. Mahmodul Hasan, to allow me to move forward with the design and development phase of this project.

Name**: Irshad Hossain**
ID: 2303030
Department of Software Engineering, UFTB
Date: 2025-09-09

_____
Signature

Contents

Document Revision History

Version	Date	Prepared By	Reviewed By	Activity
V0.1	2025-08-24	Irshad Hossain	Md. Mahmodul Hasan	Initial Proposal: Random password generator and simple password vault.
V1.0	2025-09-03	Irshad Hossain	Md. Mahmodul Hasan	Final Proposal: Complete CLI tool for developers with full functionality, requirements, attributes, monitoring, risk, and change management.

1. Executive Summary

PassGuard-CLI is a secure, terminal-based password manager designed for developers advanced users who prefer command-line interfaces. It allows users to generate random passwords, store details, and manage accounts, API keys, SSH entries, finance information, and personal notes in an encrypted local database.

Key Features:

- Fast UNIX-style subcommands.
- Interactive TUI (text-based menu mode).
- Strong encryption (AES-256).
- Security-focused features: rate-limited logins, password history, audit logs, expiry warnings.

Goal: Provide a lightweight, offline, open-source password manager seamlessly integrated into developer workflows.
Technology: C++ (modular structure), SQLite (local DB), OpenSSL (encryption).
Timeline: Design → Development → Testing → Deployment.
Risks: Security vulnerabilities (to be mitigated via testing & reviews).

2. Introduction

2.1 Purpose

Defines functional and non-functional requirements for PassGuard-CLI, ensuring a secure, efficient, and user-friendly tool for managing credentials offline.

2.2 Document Conventions

- **Bold** → Key terms/headings
- *Italic* → Emphasis, external references
- Code Blocks → CLI commands (e.g., `pass gen`)
- Numbered Lists → Ordered requirements
- Bullet Points → Features/items
- Tables → Structured data (revision history, specs)
- Dates → YYYY-MM-DD format
- Versions → Semantic (e.g., V1.0 = initial release)

2.3 Intended Audience & Reading Suggestions

- **Owner (Md. Mahmodul Hasan):** Review and approve the SRS.
- **Developers (Irshad Hossain):** Use the SRS as a reference for implementation.
- **Testers:** Verify and validate the software against the requirements.
- **Stakeholders:** Open-source community and other interested users.

2.4 Scope of Work

Core Features:

- Secure storage of Accounts, API keys, SSH, finance, personal notes.
- CLI subcommands also Interactive TUI for CRUD operations.
- AES-256 encryption with SQLite database.

Security Enhancements:

- User verification with an access key and rate-limited login attempts to prevent unauthorized access.
- Audit logs of all access attempts and expiry warnings for outdated entries.
- Automatic backups of the database for recovery.

User Experience:

- Fast commands (`pass gen --len 16 --copy`).
- Clipboard integration.

Out of Scope:

- Cloud sync, GUI, mobile/web, biometric auth.

2.4.1 Technology Specification

- **Language:** C++17+
- **Database:** SQLite
- **Encryption:** OpenSSL (AES-256)
- **Clipboard:** Platform-specific (xclip/WinAPI)
- **Build:** CMake
- **Libraries:** `std::filesystem, std::chrono, <iomanip>` (for formatted tables), termcolor / ANSI codes (for colored text), optional support for Unicode display.

Definitions:

- CLI = Command-Line Interface
- TUI = Terminal UI
- DB = Database
- AES = Advanced Encryption Standard
- API Key = Authentication key for services
- SSH = Secure Shell

Audience: Developers, advance users.

Interfaces:

- **UI:** CLI commands + TUI.
- **Hardware:** CPU, >512MB RAM, disk storage.
- **Software:** Windows

3. Specific Requirements

3.1 Development Technology

- **Frontend:** CLI and TUI in C++
- **Backend:** SQLite DB, C++ logic
- **Security:** OpenSSL AES-256
- **Build:** CMake, single binary

3.2 Functional Requirement

- **Init:** `pass init` → Initializes database and sets Access key (manual first-time setup; enforced password policy).
- **Generate:** `pass gen [flags]` → Generates random strong passwords (customizable length, character types) and optionally copies to clipboard.
- **Login:** `pass login` → Authenticate with Access key; starts session with rate-limited attempts; locks after multiple failures.
- **Add Entry:** `pass add <category> [flags]` → Add entries in categories: password, API key, SSH, finance, personal; prompts for missing details; requires login.
- **Get Entry:** `pass get <category> <service> [--copy]` → Retrieve and optionally copy entry; requires login.
- **List Entries:** `pass list <category>` → Display entries in a formatted table with details like last changed and expiry warnings; requires login.
- **Change Entry:** `pass change <category> <service> [flags]` → Update entry; old values saved in history; requires login.
- **Delete Entry:** `pass delete <category> <service>` → Soft-delete entry to recoverable trash for 7 days; requires login and confirmation.
- **Settings:** `pass settings [subcommand]` → Manage access key, backups, recoveries, expiry warnings, audit logs, and password policies; requires login.
- **Reset:** `pass reset` → Wipes database completely; requires confirmation; re-initialization needed.
- **Logout:** `pass logout` → Ends current session.
- **Interactive Mode:** Run `pass` → Open system in terminal, Menu-driven access with formatted tables, colored text, and optional Unicode display.

Categories: Account, API, SSH, Finance, Personal.

4. Non-Functional Requirements

- **Security:** Encrypted storage only.
- **Performance:** <1s for operations, support 1000+ entries.
- **Usability:** Intuitive CLI (`pass --help`).
- **Reliability:** Integrity checks + backups.
- **Portability:** Windows.
- **Maintainability:** Modular, documented.
- **Scalability:** Optimized for personal use.

5. Monitoring Strategy

- **Logging:** Audit logs stored in DB.
- **Error Handling:** Graceful input validation, DB corruption detection.
- **Sessions:** Auto-expire with warnings.
- **Testing:** Unit + integration tests.
- **Post-Deployment:** GitHub issue tracking.

6. Risk Management

- **Security Vulnerabilities:** High → Mitigate via audits & reviews.
- **DB Corruption:** Medium → Integrity checks + backups.
- **Platform Issues:** Medium → Cross-platform tests.
- **Scope Creep:** Low → Formal change process.
- **Contingency:** 20% buffer time.

7. Change Management Process

1. **Request:** Formal request (email/GitHub issue).
2. **Review:** Owner validates & approves.
3. **Documentation:** Update SRS + revision history.
4. **Implementation:** Add in next dev phase + testing.