DEPARTMENT OF CYBER SECURITY

PROJECT REPORT

Secure and Encrypted Password Manager in C++

VaultGuard++

COURSE CODE: CS112-L

SUBMITTED TO: Sheikh Qaisar Ayyub

SUBMISSION DATE: June 16, 2025

SUBMITTED BY:

Muhammad Adeel Haider	241541
Umar Farooq	241575

BS-CYS-F24-A

SECURE AND ENCRYPTED PASSWORD MANAGER IN C++

VaultGuard++

Table of Contents:

Abstract

In an age where cybersecurity threats are escalating, the protection of sensitive credentials is more important than ever. **VaultGuard++** is a secure, offline-first password manager developed in **modern C++**, designed to store user credentials with **military-grade encryption** using **Libsodium** and **OpenSSL** (AES-256, SHA-256 with salt).

This command-line application emphasizes **privacy**, **security**, and **portability**, offering features like secure CRUD operations for password entries, session management with inactivity-based auto-logout, and detailed activity logging. User-configurable settings enhance flexibility, while optional **end-to-end encrypted cloud backups** are facilitated through a Python-based integration with the **Google Drive API** using OAuth 2.0.

With a **modular architecture** consisting of components such as the Vault, Config, Timer, and Logger modules, VaultGuard++ is both maintainable and scalable. It operates fully offline by default, ensuring that users retain full control of their data without depending on external servers or services.

Introduction

In today's increasingly digital world, managing dozens of passwords across platforms has become both essential and difficult. The prevalence of weak, reused, or plaintext-stored passwords has made users vulnerable to cyberattacks such as credential stuffing, phishing, and brute-force attacks.

VaultGuard++ is a secure, offline-first password manager developed in modern C++. It is designed to protect user credentials using strong cryptographic standards, session security, and optional encrypted backups to Google Drive. By combining robust encryption with a user-friendly command-line interface, VaultGuard++ allows users to securely manage their passwords without relying on constant internet connectivity.

Problem Statement

Traditional password storage poses risks like password fatigue, where users reuse passwords, increasing vulnerability. Many tools store passwords in plaintext or use weak hashing, risking data leaks. Most password managers also rely on cloud access, raising privacy and trust concerns.

Solution Overview

VaultGuard++ addresses these problems with a zero-trust, offline-first design:

- **Offline Security**: Vaults are encrypted locally using industry-standard algorithms.
- **Optional Cloud Sync**: Users can opt to securely back up their encrypted vaults to Google Drive.
- **Zero-Knowledge Architecture**: Master passwords are never stored or transmitted in plaintext.

Project Objectives

VaultGuard++ aims to create a secure, private, and user-controllable password manager. Key objectives include:

Functional Objectives:

- Encrypt and store user credentials using modern standards.
- Provide full CRUD support via a secure command-line interface.
- Automatically log out users after inactivity to prevent session hijacking.
- Allow secure backup/restore of data via Google Drive (opt-in).
- Log all user actions (login, logout, changes) for auditing.

Performance Metrics:

- Password encryption/decryption latency: **< 50ms per entry**.
- Backup time for 100 entries: **< 10 seconds**.
- Minimal CPU/memory footprint on standard hardware.

Features Overview

Feature	Description
Master Password	Hashed using Argon2id (via PBKDF2-SHA256) with salt and 100,000 iterations.
Password Encryption	Uses XChaCha20-Poly1305 (Libsodium) for vault data.
Session Management	Auto-logout after 5 minutes of inactivity (configurable).
CRUD Operations	Add, search, update, and delete credentials.
Password Generator	Random, strong password generation (up to 64 chars).
Activity Logging	Tracks login/logout, changes, and invalid attempts.
Admin Panel	Update timeout durations and enable/disable auto-backup.
Cloud Backup	Upload encrypted `.vault` files to Google Drive via `backup.py`.

System Architecture

Frontend Layer (CLI)

- **Menu-Based Navigation**: Handles login, vault operations, admin tasks.
- **Input Validation**: Ensures numeric/valid responses and timeouts.

Core Layer

- **Vault Module**: Manages password entries, encryption, login/setup.
- **Encryption Engine**: Uses Libsodium for encrypting/decrypting entries.
- **Logger Module**: Records activity logs in `activity.vault`.
- **Timer Module**: Uses `std::chrono` for session expiry and timeouts.

Data Layer

- **Vault File (`passwords.vault`)**: Encrypted file containing credentials.
- **Config File (`vaultguard.config`)**: Stores session/backup preferences.
- **Hash File (`master.hash`)**: Securely stores master password hash.

Integration Layer

- **Cloud Backup Script (`backup.py`)**:
  - Python + Google Drive API.
  - Uploads encrypted vault file using OAuth2 credentials.

User Interface Flow (CLI)

This section outlines the step-by-step interaction of the user with the VaultGuard++ system through a Command-Line Interface (CLI).

Start Menu: [Login / Setup / Forgot / Exit]

- **Login:**
  - User is prompted for the master password.
  - Password is securely hashed and verified against the stored hash.
  - Upon success, access is granted to the main menu; failure is logged.
- **Setup:**
  - Initializes the vault if not already present.
  - Prompts user to set a strong master password.
  - Password is securely hashed and stored.
- **Forgot:**
  - Warns user that master password recovery is not possible.
  - Offers an option to delete the vault and start anew (data loss).
- **Exit:**
  - Terminates the program.

Login: Master Password Authentication

- Validates the user's identity by checking the input password against the stored hash.
- On success, decrypts and loads vault contents.
- On failure, logs the event and remains on the login screen.

Main Menu: [Password Manager / Admin Panel / Logout]

- **Password Manager:** Access core CRUD operations for password entries.
- **Admin Panel:** Configure timeout settings and backup options.
- **Logout:** Securely ends session and clears loaded data.

Password Manager: [Add / View / Delete / Search]

- **Add:** Collects entry data and adds it securely to the encrypted vault.
- **View:** Displays a list of all entries in a secure format.
- **Delete:** Allows deletion of specific entries.
- **Search:** Searches entries by service name.

Admin Panel: [Change Timeout / Toggle Backup]

- **Change Timeout:** Allows users to set inactivity timeout in minutes.
- **Toggle Backup:** Enables or disables automatic cloud backups.

Modules

Each source file contributes to the modular architecture of VaultGuard++.

main.cpp: Main Logic & Menu Handler

- Entry point of the application.
- Displays initial menus and routes user input.
- Initializes vault and session timer.

vault.cpp / vault.h: Vault Management & Authentication

- Handles vault creation and authentication.
- Implements hashing of master passwords.
- Manages vault file reading/writing.

entry.cpp / entry.h: Password Entry Definitions

- Defines the structure of each password entry.
- Manages serialization and input validation.

manager.cpp / manager.h: User Menu & CRUD Operations

- Provides interactive menu for adding, viewing, deleting, and searching entries.
- Integrates with Vault and Entry modules.

encryption.cpp / encryption.h: Encryption/Decryption Engine

- Core encryption and decryption routines using Libsodium and OpenSSL.
- Implements AES and XChaCha20-Poly1305.

timer.cpp / timer.h: Session Timeout Handling

- Tracks user inactivity using std::chrono.
- Forces logout after configured timeout.

logger.cpp / logger.h: Activity Logging System

- Logs security events such as logins, logout, CRUD actions.
- Maintains a log file with timestamps.

generator.cpp / generator.h: Secure Password Generator

- Generates strong, random passwords.
- Uses secure random functions and allows user customization.

config.cpp / config.h: Configuration Management

- Reads/writes settings like auto-logout and cloud backup preferences.
- Stores data in a simple key-value format.

backup.py: Cloud Backup Utility

- Python script for uploading encrypted vaults to Google Drive.
- Uses OAuth 2.0 and Google Drive API.

Makefile: Build Automation

- Compiles and links all source files.
- Includes targets for build, run, and clean operations.

Cloud Backup Integration

VaultGuard++ integrates with Google Drive for encrypted cloud backups using a Python script (`backup.py`). The integration relies on OAuth 2.0 for authentication, enabling secure and authorized access to the user's Google Drive account. The process differs slightly between the first run and subsequent runs.

First-Time Run

- When `backup.py` is executed for the first time, it requires the user to authenticate via Google's OAuth 2.0 consent screen.
- After the user grants access, the script receives an authorization code which it exchanges for OAuth tokens:

- The tokens are saved locally in a file (commonly named `token.pickle`) in the script directory. This token file allows the script to perform authenticated actions on behalf of the user in future runs without requiring repeated authentication.
- Once authentication is successful, the encrypted vault file (`passwords.vault`) is uploaded to the Google Drive folder `VaultGuardBackups/`. If the folder does not exist, it is created automatically.

Subsequent Runs

- For all subsequent runs of `backup.py`, the script loads the saved `token.pickle` file to authenticate with Google Drive silently.
- After successful authentication, the script uploads or updates the encrypted vault file on Google Drive.

This design balances security and convenience, allowing VaultGuard++ to provide secure cloud backup capabilities without compromising offline-first principles or requiring repeated user authentication.

Build and Run Instructions

This section provides step-by-step guidance on how to build, run, and utilize VaultGuard++

Technologies Used

Layer	Technology
Core Backend	C++17 (Vault logic, CLI, file management)
Encryption	Libsodium (XChaCha20-Poly1305, Argon2id)
Hashing	OpenSSL (SHA-256, PBKDF2)
Backup Integration	Python (Google Drive API + OAuth 2.0)
Build & Platform	Makefile

Prerequisites

- **OS**: Windows 10+
- **Compiler**: MinGW (mingw32-make) or MSVC (C++17)
- **Libraries**: Libsodium, OpenSSL
- **Python 3.x** with packages:
  `google-api-python-client`, `google-auth-httplib2`, `google-auth-oauthlib`, `cryptography`
- **Install Python packages**:

```
pip install google-api-python-client google-auth-httplib2 google-auth-oauthlib cryptography
```

Build and Run

1. Clone the repo and open terminal:
   `cd vaultguard++`
2. Run build command:
   `mingw32-make`
3. Run the app:
   `./bin/vaultguard.exe`

Security Considerations

VaultGuard++ is designed with a strong emphasis on security, considering the sensitive nature of password management. The following security practices and features are implemented to safeguard user data and maintain confidentiality:

- **Master Password Protection**
  The master password is never stored in plaintext. It undergoes hashing using **Argon2id**, a memory-hard key derivation function resistant to brute-force and side-channel attacks. This ensures that even if the vault data is compromised, attackers cannot easily recover the master password.
- **Data Encryption**
  All password entries and vault contents are encrypted using **Libsodium's XChaCha20-Poly1305** authenticated encryption algorithm. This provides confidentiality, integrity, and authenticity, ensuring that stored credentials cannot be tampered with or read by unauthorized parties.
- **Zero-Knowledge Design**
  VaultGuard++ operates on an offline-first, zero-knowledge principle — all encryption and decryption happen locally. The master password and decrypted data never leave the device, reducing exposure to network-based attacks.
- **Secure Cloud Backups**
  The cloud backup integration uploads only encrypted vault files. Before syncing, the vault data is encrypted locally; the cloud never sees unencrypted passwords. OAuth 2.0 authentication with token caching adds an additional layer of access control to cloud resources.
- **Session Management and Timeouts**
  To prevent unauthorized access, VaultGuard++ implements auto-logout features that lock the vault after a configurable period of inactivity. Input timeouts on password prompts minimize the risk of shoulder surfing and automated brute-force attempts.
- **Activity Logging and Audit Trails**
  All critical actions such as logins, password modifications, and failed access attempts are logged securely. These logs help detect suspicious behavior or unauthorized access and assist in forensic investigations.
- **Input Validation and Sanitization**
  The CLI interface validates user input rigorously to prevent injection attacks or buffer overflow vulnerabilities, which are common pitfalls in C++ applications.

- **Cross-Platform Consistency**
  The cryptographic primitives and file handling are tested and consistent across supported platforms (Windows and Linux), reducing platform-specific security loopholes.

Limitations and Future Enhancements

While VaultGuard++ offers a robust and secure offline password management system with encrypted cloud backup capabilities, there are several areas identified for improvement and expansion in future versions:

- **Lack of Graphical User Interface (GUI)**
  Currently, VaultGuard++ operates exclusively via the command-line interface (CLI), which may not be user-friendly for all users. Developing a GUI would improve accessibility, ease of use, and broaden the user base.
- **Single-User Architecture**
  The system is designed for individual users only. Multi-user support with role-based access controls and shared vault capabilities would enhance its applicability in team or enterprise environments.
- **No Biometric or Two-Factor Authentication (2FA)**
  Adding support for biometric login (fingerprint, facial recognition) or integrating 2FA methods would strengthen security, especially against stolen device scenarios.
- **Enhanced Logging and Alerts**
  Introducing real-time alerting mechanisms or integration with security monitoring tools can provide proactive security management.
- **Cross-Platform Installer and Auto-Update**
  Simplifying deployment through dedicated installers and providing automatic update mechanisms would improve user onboarding and maintenance.

Conclusion

VaultGuard++ offers a secure and privacy-focused solution to password management, ideal for users who value offline protection with the option for encrypted cloud backup. Developed in modern C++ with trusted libraries like Libsodium and OpenSSL, it ensures that user data remains safe and tamper-proof. By addressing key issues such as password fatigue, weak storage methods, and cloud dependency, VaultGuard++ stands out with its offline-first, zero-knowledge design. While it currently has limitations like a command-line interface and no multi-user support, it sets a solid groundwork for future improvements. Overall, VaultGuard++ demonstrates how strong security and usability can be combined in a modern, scalable password management tool.

The contributions by the development team — Muhammad Adeel Haider and Umar Farooq from Air University Cyber Security department (BSCYS-F24-A) — demonstrate practical application of cryptographic principles and software engineering best practices, aiming to empower users with greater control over their digital identities.