Software Requirements Specification (SRS) for Password Manager

Project Title: Password Manager (Python-based with GUI)

Group Members:

- **M.Tamseel Khanzada**

  Roll No: 23k-2063
- **Abdul Ahad**

  Roll No: 23p – 0625
- **Abdul Samad**

  Roll No: 23k-3042

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements and specifications for the development of a **Password Manager** application. This application will securely store and manage user passwords, providing encryption, authentication, and an intuitive interface to safeguard sensitive credentials.

1.2 Scope

The Password Manager will allow users to create an account, log in securely, store multiple sets of credentials for different websites, retrieve them when needed, and manage (add, update, delete) stored passwords. Data security is ensured through encryption techniques such as AES-GCM.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **AES**: Advanced Encryption Standard
- **GCM**: Galois/Counter Mode
- **IV**: Initialization Vector
- **MFA**: Multi-Factor Authentication

1.4 References

- NIST AES Specifications
- OWASP Guidelines for Secure Password Storage
- Python Documentation for Cryptography

2. Overall Description

2.1 Product Perspective

The Password Manager will function as a **standalone desktop application** initially. Future versions may incorporate cloud synchronization and browser extensions. It is being built with Python and will use JSON files for data storage in the local environment.

2.2 Product Functions

- User signup and login
- Secure password storage with encryption
- Password retrieval with decryption
- Account profile management

- Support for multiple website entries
- Data integrity checks
- Basic password generation tool

2.3 User Characteristics

Target users are individuals who want to securely manage their passwords without relying on online third-party services. They are expected to have basic knowledge of computer use but no technical expertise in cybersecurity.

2.4 Constraints

- Application must operate offline.
- Data is stored locally; loss of the device may lead to loss of data unless backed up manually.
- Cryptography libraries used must comply with strong encryption standards.

2.5 Assumptions and Dependencies

- Python 3.10 or higher must be installed.
- The user is responsible for backing up their data.
- System must have cryptography libraries installed (e.g., `cryptography` package).

3. Specific Requirements

3.1 Functional Requirements

- **Account Creation**: Allow users to sign up with a master password.
- **Login**: Authenticate users before granting access.
- **Password Storage**: Encrypt and store each password entry individually.
- **Password Retrieval**: Decrypt and display password upon correct authentication.
- **Password Management**: Enable users to add, edit, and delete saved credentials.
- **Password Search**: Implement a search functionality to quickly locate stored entries.

3.2 Non-Functional Requirements

- **Security**: All passwords must be encrypted using AES-GCM.
- **Performance**: Password lookup and decryption must occur within 1 second.
- **Usability**: The application must have a simple and clean interface.
- **Reliability**: System should not crash under normal usage conditions.
- **Scalability**: Designed in a modular way to allow future cloud sync features.

3.3 External Interface Requirements

- **User Interface**: Built using Tkinter for a desktop application.
- **Data Storage Interface**: Read/write encrypted data to a JSON file.
- **Encryption Interface**: Utilize `cryptography` library APIs.

4. System Features

4.1 User Authentication

- Sign-up and login using master password.
- Master password is securely hashed and not stored in plaintext.
- Optional integration of MFA (e.g., OTP code) for additional security.

4.2 Password Encryption/Decryption

- Each password entry is encrypted with a unique key/IV combination.
- Encryption process is transparent to the user.
- Decryption happens only in memory; decrypted passwords are never saved.

4.3 Password Management

- Add, view, edit, and delete credentials for various websites.
- Automatic timestamp generation for when credentials are added/modified.
- Ability to generate random strong passwords using built-in generator.

4.4 Data Backup and Restore

- Export and import encrypted JSON file manually.

- Optional password-protected backup file creation.

5. Other Requirements

5.1 Security Requirements

- All cryptographic materials (keys, IVs) are randomly generated.
- Data is encrypted at rest and never stored in plaintext.
- Implement session timeouts after a period of inactivity.

5.2 Software Quality Attributes

- **Availability**: 99.9% during active user sessions.
- **Maintainability**: Modular and documented codebase.
- **Portability**: Should run on Windows, Linux, and macOS systems.

5.3 Business Rules

- Users are responsible for remembering their master password.
- Forgotten master password results in **complete data loss** as no recovery option is provided (for security reasons).

Appendix A: Glossary

- **SRS**: Software Requirements Specification — A document that describes the software system to be developed, outlining its requirements and functionality.
- **Password Manager**: A software application used for securely storing and managing a user's passwords and other sensitive information.
- **AES-GCM**: Advanced Encryption Standard - Galois/Counter Mode; an encryption algorithm providing confidentiality and data authenticity.
- **Encryption**: The process of encoding data to prevent unauthorized access.
- **Authentication**: The process of verifying the identity of a user or system, typically involving the use of credentials such as usernames and passwords.

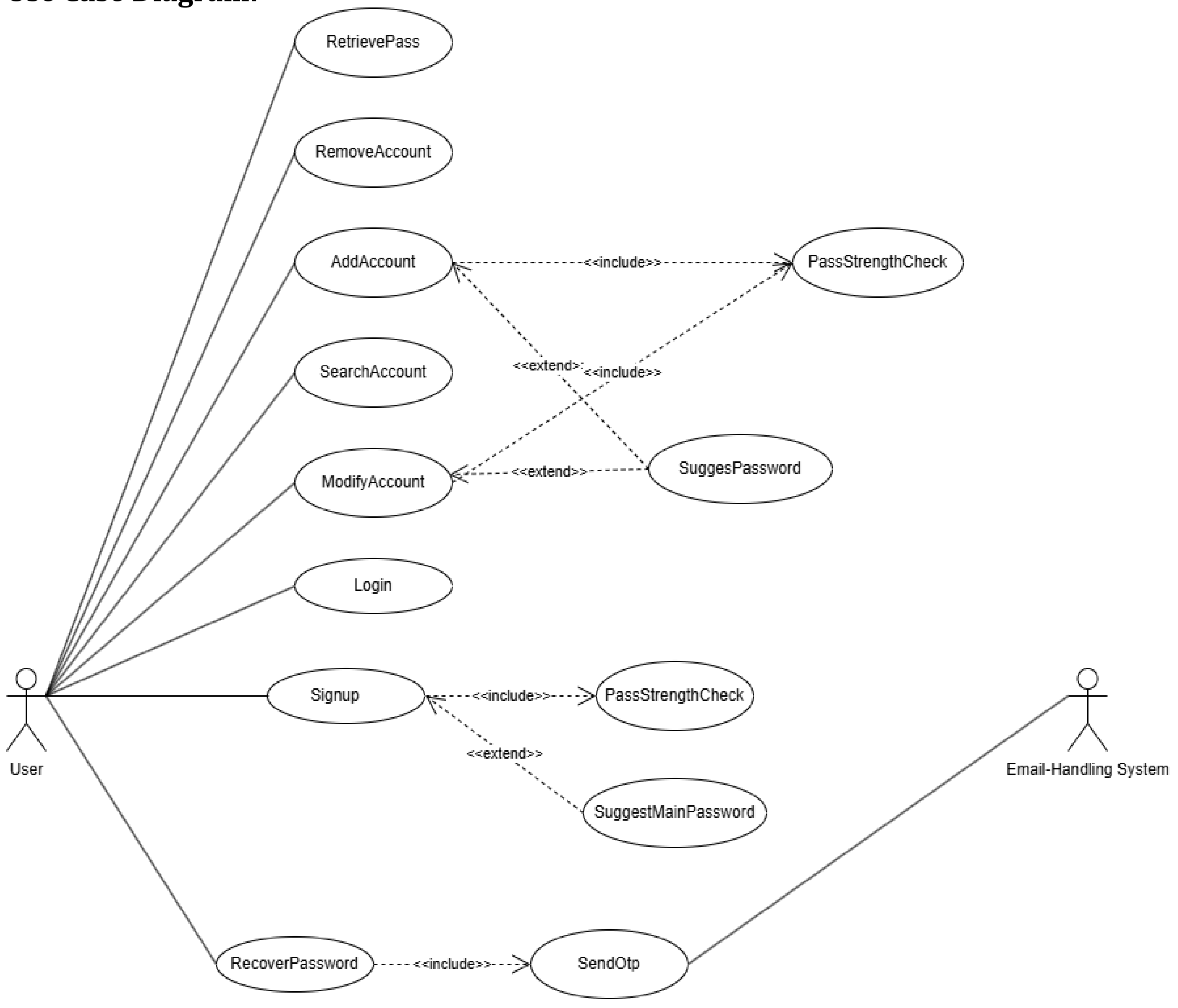
- **Master Password**: A primary password that grants access to all stored passwords within the password manager.
- **JSON**: JavaScript Object Notation — A lightweight data-interchange format used to store encrypted password entries.
- **Use Case Diagram**: A visual representation of the system's functionality from the user's perspective, illustrating interactions between the user and the system.
- **MFA**: Multi-Factor Authentication — A security system requiring more than one method of authentication for enhanced security.
- **Base64 Encoding**: A method of encoding binary data into ASCII characters to facilitate safe storage and transmission.

Appendix B: Analysis Models

The **Use Case Diagram** below illustrates the interactions between the user and the system:

- **Actors**:
  - User
  - Email Handling System
- **Use Cases**:
  - Sign up
  - Retrieve Password
  - Remove Account
  - Add Account
  - Search Account
  - Modify Account
  - Login
  - Recover Password
  - Password Strength Check
  - Suggest Password
  - Send OTP

Use Case Diagram:



End of Document