



Faculty of Computer Science and Information Technology

***AN OFFLINE PASSWORD MANAGEMENT SYSTEM WITH PERFORMANCE AND
SECURITY ANALYSIS ON AES ENCRYPTION AND PBKDF2***

Raphael Kong Chee Hwen

Bachelor of Computer Science with Honours

(Network Computing)

2025

**AN OFFLINE PASSWORD MANAGEMENT SYSTEM WITH PERFORMANCE AND
SECURITY ANALYSIS ON AES ENCRYPTION AND PBKDF2**

RAPHAEL KONG CHEE HWEN

This project is submitted in partial fulfilment of the
requirements for the degree of
Bachelor of Computer Science with Honours
(Network Computing)

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK

2025

**SISTEM PENGURUSAN KATA LALUAN TANPA TALIAN DENGAN ANALISIS
PRESTASI DAN KESELAMATAN PADA ENKRIPSI AES DAN PBKDF2**

RAPHAEL KONG CHEE HWEN

Projek ini merupakan salah satu keperluan untuk
Ijazah Sarjana Muda Sains Komputer dengan Kepujian
(Pengkomputeran Rangkaian)

Fakulti Sains Komputer dan Teknologi Maklumat

UNIVERSITI MALAYSIA SARAWAK

2025

DECLARATION

I hereby declare that this project entitled “An Offline Password Management System with Performance and Security Analysis on AES Encryption and PBKDF2” is the result of my own efforts. All the information reference from other sources had been fully acknowledged. I fully understand that any plagiarism or cheating would be subjected to disciplinary actions.

Signature : 

Name : RAPHAEL KONG CHEE HWEN

Date : 20/6/2025

ACKNOWLEDGEMENT

Firstly, I would like to extend my heartfelt gratitude and sincere appreciation to my supervisor, Dr. Ling Yeong Tyng, for her invaluable and constructive suggestions, guidance, and encouragement throughout the duration of this project. Her generous willingness to dedicate her time and effort has been deeply appreciated. This project would not have been possible without her unwavering support and insightful advice.

I would also like to express my deepest thanks to my family for their unconditional support, understanding, and love at every step of this journey. Their belief in me has been a constant source of strength and motivation.

TABLE OF CONTENTS

DECLARATION	i
ACKNOWLEDGEMENT	ii
TABLE OF CONTENTS	iii
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
LIST OF ABBREVIATIONS.....	x
ABSTRACT	xi
ABSTRAK	xii
CHAPTER 1: INTRODUCTION	1
1.1 Overview	1
1.2 Background	1
1.3 Problem Statement	3
1.4 Objectives.....	4
1.5 Methodology	5
1.5.1 Requirement Analysis	6
1.5.2 System Design	7
1.5.3 Implementation	8
1.5.4 Testing.....	9
1.5.5 Deployment.....	10
1.5.6 Maintenance	10
1.6 Scope	10
1.7 Significance of Project	13
1.8 Project Schedule.....	14
1.9 Expected Outcome	15
1.10 Summary	15
CHAPTER 2: LITERATURE REVIEW	16
2.1 Overview	16
2.2 Encryption Methods: AES and PBKDF2	16
2.2.1 Encryption	16
2.2.2 AES (Advanced Encryption Standard) Encryption.....	17
2.2.3 PBKDF2 (Password-Based Key Derivation Function 2).....	18
2.2.4 Integration of AES and PBKDF2 in Password Encryption.....	18
2.3 Review on Similar Existing System.....	19

2.3.1	1Password.....	19
2.3.2	Keeper	23
2.3.3	Total Password	27
2.3.4	Sticky Password	30
2.4	Comparison of Features between Existing Systems and Proposed System	33
2.5	Brief Overview of Proposed System	35
2.6	Review of Tools and Technology Used.....	36
2.7	Summary	38
CHAPTER 3: REQUIREMENT ANALYSIS & DESIGN		39
3.1	Overview	39
3.2	Requirement Analysis	39
3.2.1	Method for Gathering Requirements.....	39
3.2.2	Software Requirement Specification.....	46
3.2.3	Hardware Requirement Specification	46
3.3	System Design.....	47
3.3.1	Use Case Diagram.....	47
3.3.2	Sequence Diagram	49
3.3.3	Class Diagram.....	56
3.3.4	Flowchart	58
3.3.5	User Interface Design	60
3.4	Summary	65
CHAPTER 4: IMPLEMENTATION.....		67
4.1	Overview	67
4.2	Implementation of the Password Management System.....	67
4.2.1	Hardware Specification.....	67
4.2.2	Integrated Development Environment (IDE).....	68
4.2.3	Database Setup.....	71
4.2.4	Library Installation.....	74
4.2.5	Coding.....	74
4.3	The Developed System	94
4.3.1	Login Screen	94
4.3.2	Main Menu.....	96
4.3.3	Add/Edit Password Entries	97
4.3.4	Password Generation	99
4.3.5	Password Breach Check.....	101
4.3.6	Password Strength Tester Menu.....	102

4.3.8 AES Bit Evaluation Menu.....	104
4.3.9 Guess Configuration Menu	106
4.3.10 Password Health Checker Menu	108
4.3.11 Settings Menu	111
4.4 Summary.....	112
CHAPTER 5: TESTING	113
5.1 Overview	113
5.2 Functional Testing	113
5.2.1 Unit Testing.....	113
5.2.2 User Acceptance Testing.....	123
5.3 Non-Functional Testing	133
5.3.1 Performance Testing	133
5.3.2 Security Testing.....	143
5.4 Summary.....	148
CHAPTER 6: CONCLUSION AND FUTURE WORKS	149
6.1 Overview	149
6.2 Achievements.....	149
6.3 Limitations and Constraints.....	151
6.4 Future Works	152
6.5 Summary.....	153
REFERENCES.....	154
APPENDICES.....	159
APPENDIX A: SURVEY ON PASSWORD MANAGEMENT PRACTICES	159
APPENDIX B: CODE SNIPPET FOR PASSWORD GENERATION.....	167
APPENDIX C: CODE SNIPPET FOR ESTIMATING PASSWORD CRACK TIME	169
APPENDIX D: CODE SNIPPET FOR ACTUAL DICTIONARY ATTACK TIME.....	171
APPENDIX E: CODE SNIPPET FOR ACTUAL RAINBOW TABLE ATTACK TIME .	173
APPENDIX F: AES BIT PERFORMANCE METRICS	175
APPENDIX G: AES BIT SECURITY METRICS	177
APPENDIX H: DETERMINING THE BEST AES BIT SIZE	178
APPENDIX I: USER ACCEPTANCE TESTING SURVEY FOR THE DEVELOPED PASSWORD MANAGEMENT SYSTEM	180

LIST OF TABLES

Table 1.1 Scope	10
Table 2.1 Comparison Table of Existing System and Proposed Prototype	33
Table 3.1 List of Software	46
Table 3.2 List of Hardware.....	47
Table 5.1 User Registration Function Test Case	114
Table 5.2 User Login Function Test Case	114
Table 5.3 OTP Verification Test Case.....	116
Table 5.4 Password Encryption and Decryption Test Case.....	117
Table 5.5 Password Generation Test Case.....	118
Table 5.6 Password Breach Check Test Case	119
Table 5.7 Password Strength Testing Test Case	120
Table 5.8 AES Bit Evaluation Test Case	122
Table 5.9 Performance Testing for Password Management System	134
Table 5.10 Estimated Time to Crack	144
Table 5.11 Actual Attack Time	146
Table 6.1 Objectives and Achievements.....	150

LIST OF FIGURES

Figure 1.1 Waterfall Model	6
Figure 1.2 Gantt Chart for Final Year Project (FYP) 1	14
Table 2.1 Comparison Table of Existing System and Proposed Prototype	33
Figure 2.1 AES Encryption and Decryption Process	17
Figure 2.2 Main Menu for 1Password.....	20
Figure 2.3 Add Item Options in 1Password	21
Figure 2.4 Dashboard for Watchtower and Password Health Checker	22
Figure 2.5 Main Menu for Keeper	24
Figure 2.6 Add Item Option in Keeper.....	25
Figure 2.7 Auto-fill Feature in Keeper	26
Figure 2.8 Main Menu for Total Password.....	28
Figure 2.9 Security Report in Total Password.....	29
Figure 2.10 Main Menu for Total Password.....	30
Figure 2.11 Password Health Checker for Stick Password	31
Figure 3.1 Concerns on Password Management Practices.....	40
Figure 3.2 Frequency of Password Reuse Across Multiple Accounts.....	41
Figure 3.3 Preference to Use a Digital Password Manager.....	42
Figure 3.4 Respondent Trust in the Effectiveness of Digital Password Managers	43
Figure 3.5 Respondent Feature Considerations in Selecting a Password Management System	44
Figure 3.6 Challenges and Concerns Preventing Adoption of Password Management Systems	45
Figure 3.7 Use Case Diagram of Password Management System	48
Figure 3.8 Sequence Diagram for Registration Process.....	49
Figure 3.9 Sequence Diagram for Login Process.....	50
Figure 3.10 Sequence Diagram for Manage Passwords (Add/Edit/Delete) Process.....	51
Figure 3.11 Sequence Diagram for View Password Health Process	52
Figure 3.12 Sequence Diagram for Generate Password Process	53
Figure 3.13 Sequence Diagram for Configure Setting (MFA with OTP) Process	54
Figure 3.14 Sequence Diagram for Configure Other Settings Process	55

Figure 3.15 Class Diagram.....	57
Figure 3.16 Flowchart of How Passwords are Securely Stored and Retrieved.....	59
Figure 3.17 Main Menu for Password Management System Prototype	61
Figure 3.18 Add Entry Form	62
Figure 3.19 Edit Entry Form	63
Figure 3.20 Password Health Checker Menu.....	64
Figure 3.21 Settings Menu	65
Figure 4.1 Visual Studio Code Interface for System Development	68
Figure 4.2 Python Extension in the VS Code Marketplace.....	70
Figure 4.3 “passwords.db” File in DB Browser for SQLite.....	71
Figure 4.4 Code Snippet for Obfuscation of the Database Password	75
Figure 4.5 Master Password Setup and Key Derivation	77
Figure 4.6 Verifying Master Password during Login.....	78
Figure 4.7 Password Encryption and Decryption Using AES.....	79
Figure 4.8 Generating Recovery Keys	80
Figure 4.9 Verifying Recovery Key During Login	81
Figure 4.10 Generating OTP and Displaying QR Code	82
Figure 4.11 Verifying OTP During Login	83
Figure 4.12 Password Breach Check.....	85
Figure 4.13 Measuring Guess Rate on User’s Device.....	88
Figure 4.14 Actual Password Brute Force Attack Simulation.....	90
Figure 4.15 Actual AES Brute Force Attack Simulation.....	91
Figure 4.16 Login Screen	94
Figure 4.17 Login Screen	95
Figure 4.18 Main Menu.....	96
Figure 4.19 Add New Password Entry Form	97
Figure 4.20 Edit Password Entry Form.....	98
Figure 4.21 Context Menu for Generate Password Button	99
Figure 4.22 Password Generation Configuration Menu.....	100
Figure 4.23 Password Breached.....	101
Figure 4.24 Password Not Breached	101
Figure 4.25 Password Strength Test Button	102

Figure 4.26 Password Strength Tester Menu.....	103
Figure 4.27 "Pick For Me" Button	104
Figure 4.28 AES Bit Evaluation Menu.....	105
Figure 4.29 Guess Per Second Button.....	106
Figure 4.30 Guess Configuration Menu	107
Figure 4.31 Password Health Checker Menu	108
Figure 4.32 Password with Issues Selected.....	109
Figure 4.33 Normal Navigation Bar.....	110
Figure 4.34 Navigation Bar with Password Issues.....	110
Figure 4.35 Settings Menu	111

LIST OF ABBREVIATIONS

<i>AES</i>	Advanced Encryption Standard
<i>FYP1</i>	Final Year Project 1
<i>MFA</i>	Multi-Factor Authentication
<i>OTP</i>	One-Time Password
<i>PBKDF2</i>	Password-Based Key Derivation Function 2
<i>SQL</i>	Structured Query Language

ABSTRACT

With the rise of online accounts, users increasingly face challenges in managing secure passwords, often resorting to weak, reused, or already compromised credentials—exposing them to data breaches and credential-stuffing attacks. Most password managers rely on a single master password, creating a critical single point of failure if it is lost or compromised. Additionally, users are typically provided with vague password strength indicators that lack insight into how passwords would perform under real-world attack conditions such as plaintext brute-force, AES brute-force, dictionary, and rainbow table attacks. This project aims to enhance password security, reduce reliance on single-point authentication, and offer transparent password strength evaluation through an offline password management system. The system uses AES encryption and PBKDF2 hashing to secure the vault and strengthen key derivation. It integrates a built-in password generator and a breach check feature powered by the Pwned Passwords API to help users create strong, unique passwords and detect compromised ones. To mitigate the risks of master password dependency, the system includes multi-factor authentication (MFA) and recovery keys for secure access and recovery. A standout feature is the password strength tester, which simulates various attack methods—including plaintext brute-force, AES brute-force, dictionary, and rainbow table attacks—to evaluate each password's resilience, presenting both estimated and actual time to crack. The system was developed using the Waterfall model, a structured, phase-by-phase methodology that guided the process from requirement analysis through design, implementation, testing, and evaluation across various user scenarios, with a focus on encryption performance and attack simulations. Results demonstrate that the system successfully achieves its objectives: it improves password quality through generation and breach detection, reduces vulnerability through MFA and recovery mechanisms, and enhances user awareness by providing meaningful feedback on password strength under realistic attack models. The real-time simulation and performance analysis enable users to understand their password security level and adjust accordingly. By bridging the gap between usability and cybersecurity, this system not only secures password storage but also educates users on best practices and threat resilience.

ABSTRAK

Dengan peningkatan penggunaan akaun dalam talian, pengguna semakin menghadapi cabaran dalam mengurus kata laluan yang selamat, sering kali bergantung kepada kata laluan yang lemah, digunakan semula, atau telahpun dikompromi—mendedahkan mereka kepada kebocoran data dan serangan "credential stuffing". Kebanyakan pengurus kata laluan bergantung kepada satu kata laluan induk, yang mewujudkan titik kegagalan tunggal yang kritikal sekiranya ia hilang atau dikompromi. Selain itu, pengguna biasanya diberikan penunjuk kekuatan kata laluan yang kabur dan tidak memberikan gambaran sebenar tentang sejauh mana ketahanan kata laluan terhadap serangan dunia sebenar seperti serangan brute-force plaintext, brute-force AES, serangan kamus, dan serangan jadual pelangi. Projek ini bertujuan untuk meningkatkan keselamatan kata laluan, mengurangkan kebergantungan kepada pengesahan satu titik, dan menawarkan penilaian kekuatan kata laluan yang telus melalui sistem pengurusan kata laluan luar talian. Sistem ini menggunakan penyulitan AES dan fungsi hashing PBKDF2 untuk melindungi peti simpan kata laluan serta mengukuhkan proses penjanaan kunci. Ia menggabungkan penjana kata laluan terbina dalam dan ciri semakan pelanggaran yang menggunakan API Pwned Passwords untuk membantu pengguna mencipta kata laluan yang kukuh dan unik serta mengesan jika kata laluan mereka telah terdedah. Bagi mengurangkan risiko yang dikaitkan dengan kebergantungan kepada kata laluan induk, sistem ini turut menyertakan pengesahan pelbagai faktor (MFA) dan kunci pemulihan bagi memastikan akses dan pemulihan yang selamat. Ciri utama sistem ini ialah penguji kekuatan kata laluan, yang mensimulasikan pelbagai kaedah serangan—termasuk brute-force plaintext, brute-force AES, serangan kamus, dan serangan jadual pelangi—bagi menilai ketahanan setiap kata laluan, dengan menyampaikan anggaran masa serta masa sebenar yang diperlukan untuk memecahkan kata laluan tersebut. Sistem ini dibangunkan menggunakan model Waterfall, iaitu satu metodologi berstruktur yang berasaskan fasa demi fasa, yang membimbing proses dari analisis keperluan hingga ke reka bentuk, pelaksanaan, pengujian, dan penilaian merentasi pelbagai senario pengguna, dengan penekanan kepada prestasi penyulitan dan simulasi serangan. Hasilnya menunjukkan bahawa sistem ini berjaya mencapai objektifnya: ia meningkatkan kualiti kata laluan melalui penjanaan dan pengesanan pelanggaran, mengurangkan kerentanan melalui MFA dan mekanisme pemulihan, serta meningkatkan kesedaran pengguna dengan memberikan maklum balas bermakna tentang kekuatan kata laluan berdasarkan model serangan yang realistik. Simulasi masa nyata dan analisis prestasi ini membolehkan pengguna memahami tahap keselamatan kata laluan mereka dan membuat penyesuaian yang sewajarnya. Dengan merapatkan jurang antara kebolegunaan dan keselamatan siber, sistem ini bukan sahaja memastikan penyimpanan kata laluan yang selamat, tetapi juga mendidik pengguna tentang amalan terbaik dan ketahanan terhadap ancaman.

CHAPTER 1: INTRODUCTION

1.1 Overview

This chapter provides an introduction to the project by outlining its background, the identified problems that necessitated its development, and the objectives it aims to achieve. It also discusses the methodology employed in the project, the scope of work, and the project schedule. Additionally, the chapter highlights the significance of the project, the expected outcomes, and how these components contribute to addressing the identified problems and meeting the project's goals.

1.2 Background

In the digital era, managing passwords has become increasingly complex. With numerous online accounts ranging from social media to banking, many users face the challenge of remembering strong and unique passwords for each account. Unfortunately, this often leads to poor security practices, such as reusing the same password across multiple platforms, which significantly increases vulnerability to cyberattacks (Judge, 2022). Additionally, users may remain unaware when their passwords are compromised in data breaches, further exposing them to risks.

One common weakness in traditional password managers is the reliance on a master password. This creates a single point of failure—if the master password is lost or compromised, all associated accounts are at risk (Fernando et al., 2023). The increasing sophistication of cyber threats highlights the need for a more secure and robust password management solution.

However, lack of transparency in password strength evaluation remains another challenge. Most password management systems provide vague or generic strength indicators,

leaving users with no clear visibility into how secure their passwords truly are (Atzori et al., 2024). These systems often fail to simulate real-world attacks, such as plaintext brute-force, AES brute-force, dictionary attacks, and rainbow table attacks. As a result, users may be unaware of their passwords' vulnerabilities, and without actionable feedback, they are unable to strengthen their passwords effectively.

This password management system is designed to address these critical issues by providing transparent password strength evaluation through simulated attacks. Unlike traditional systems, it will simulate various attack methods, including plaintext brute-force, AES brute-force, dictionary attacks, and rainbow table attacks. For each method, the system will provide both an estimated time to crack the password and a real-time simulation of the attack process. This allows users to comprehensively understand their password's resilience and provides actionable feedback for improvement.

The system also uses advanced encryption methods such as AES-256 (Advanced Encryption Standard) and PBKDF2 (Password-Based Key Derivation Function 2), to securely store and protect passwords, ensuring that user data remains safe from unauthorized access (Khande et al., 2021). AES-256 is widely regarded as one of the most secure encryption standards, providing 256-bit encryption that resists brute-force decryption attempts (Alenezi et al., 2020). Recognized by the National Institute of Standards and Technology (NIST), AES-256 is commonly used for safeguarding highly sensitive data, including government-classified information (Smid, 2021).

Furthermore, PBKDF2 enhances the security of passwords by adding computational complexity. It applies multiple iterations of hashing, making the system more resistant to common attacks like brute-force and dictionary-based methods (Mustafa, 2024). The

combination of AES-256 and PBKDF2 ensures that this system offers strong encryption while maintaining efficient performance and usability.

Additionally, the system includes a built-in password generator to help users create strong, unique passwords, reducing the risk of weak or reused passwords (Rahalkar & Gujar, 2019). It also includes a breach check function that uses the Pwned Passwords API¹. This function alerts users if their passwords have appeared in known data breaches, enabling them to take immediate action to change compromised passwords and reduce the risk of unauthorized access to their accounts. The system further mitigates the risks associated with traditional password managers by incorporating multi-factor authentication (MFA) and recovery keys. These features ensure that the password vault can be securely recovered if the master password is lost or compromised, providing additional protection for sensitive user data.

1.3 Problem Statement

With the rise of online services, individuals find themselves juggling numerous accounts that demand strong password management. However, many users face several challenges in maintaining strong password practices:

- a) **Weak, Reused, and Compromised Passwords:** Users often create weak or reused passwords across multiple accounts, significantly increasing their vulnerability to hacking, data breaches, and credential stuffing attacks. Additionally, many users are unaware when their passwords have been compromised in known data breaches, leaving their accounts exposed without their knowledge.

¹ Pwned Passwords API. *GitHub*. <https://github.com/lionheart/pwnedpasswords>

- b) **Single Point of Failure:** Most password managers rely on a master password, which creates a single point of failure. If the master password is compromised or forgotten, the entire password vault is vulnerable, leaving sensitive information exposed and inaccessible.
- c) **Lack of Transparency in Password Strength Evaluation:** Users often have no clear visibility into how secure their passwords truly are. Most systems provide vague or generic strength indicators without showing how passwords would hold up against real-world attacks such as plaintext brute-force, AES brute-force, dictionary attacks, and rainbow table attacks.

1.4 Objectives

The objectives of this project are designed to directly address the challenges outlined in the problem statement and provide effective solutions for managing passwords securely and conveniently. These objectives include:

- a) **Enhancing Password Security:** To tackle the issue of weak, reused, and compromised passwords, the system will include a built-in password generator for creating strong, unique passwords and a breach check function that uses the Pwned Passwords API¹ to alert users if their passwords have appeared in known data breaches.
- b) **Mitigating Single Point of Failure:** To resolve the vulnerability posed by the reliance on a master password, the system will incorporate multi-factor authentication (MFA) and recovery keys, enabling secure recovery of the password vault even if the master password is forgotten or compromised.

c) Providing Transparent Password Strength Evaluation Through Simulated Attacks:

To help users understand the actual strength of their passwords, the system will incorporate a feature that simulates multiple attack methods, including plaintext password brute-force, AES brute-force, dictionary attacks, and rainbow table attacks. For each method, the system will provide both an estimated time to crack the password and a real-time simulation of the attack process. This will provide users with a comprehensive understanding of their password's resilience and offer actionable feedback for improvement.

1.5 Methodology

The development of this system will follow the Waterfall Model (Pargaonkar, 2023). Figure 1.1 illustrates the Waterfall Model, a linear and sequential software development methodology, where each phase flows downwards to the next, resembling a waterfall. It consists of several stages: requirement analysis, system design, implementation, testing, deployment, and maintenance. Each phase must be completed before moving to the next, with little overlap or iteration between them. This method is ideal for this system because it ensures that all the system's requirements and design details are well understood and documented before any coding starts. Since this system deals with sensitive data, it is crucial to carefully analyze, design, and test the system to protect users' information. The Waterfall Model provides clear checkpoints that help reduce risks and ensure everything runs smoothly. Its straightforward process also creates solid documentation, which is essential for keeping the system secure and easy to maintain after it's launched.

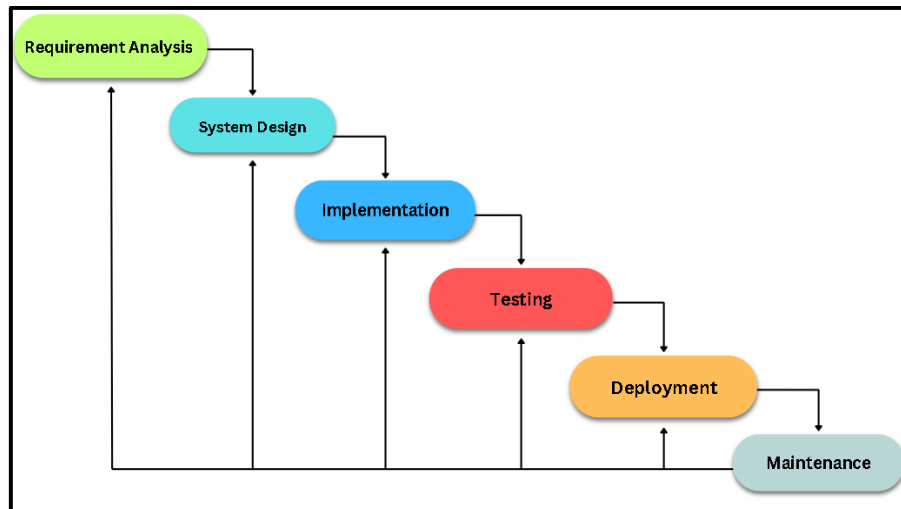


Figure 1.1 Waterfall Model

Note. The figure presents the Waterfall Model, a linear software development approach that progresses through six key phases: Requirement Analysis, System Design, Implementation, Testing, Deployment, and Maintenance. This model emphasizes structured planning and documentation, making it ideal for projects with stable requirements.

1.5.1 Requirement Analysis

During this phase, the focus is on identifying and defining the features required for the password management system. This process involves gathering user data through a survey conducted with a diverse group of respondents, which helps inform the development of system specifications. Key findings indicate that users prioritize security, convenience in password management, and concerns about privacy. The results emphasize the demand for password management systems with features like multi-factor authentication, recovery keys, and backup options. The phase concludes with the creation of detailed documentation that outlines the system's functionality, such as secure password storage using AES-256 encryption and PBKDF2 hashing.

1.5.2 System Design

The system design phase outlines the structure and functionality of the password management system through various diagrams and the user interface design. This includes:

- **Use Case Diagram:** Defines the system's functional requirements by illustrating interactions between users and the system, highlighting key actions such as registering, logging in, managing passwords (add/edit/delete), viewing password health, generating passwords, and configuring system settings.
- **Sequence Diagram:** Provides detailed step-by-step processes for each key action from the use case diagram, showing the flow of interactions between components such as the user, the password management system, the database, and the OTP (One-Time Password) application.
- **Class Diagram:** Details the system's classes and their relationships, providing a clear structure for the organization and management of data, such as user credentials and encrypted passwords.
- **Flowchart:** Demonstrates the step-by-step process for securely storing and retrieving passwords using AES-256 encryption and PBKDF2 hashing, ensuring data protection at every stage.
- **User Interface Design:** Includes interfaces created with Figma to visualize the system's layout, featuring the main menu, add/edit password forms, settings menu, and password health checker, prioritizing ease of use and intuitive navigation.

1.5.3 Implementation

The implementation phase primarily involves coding the system in Python, which focuses on building a secure and efficient password management system. This section provides a basic overview of the system's key modules, with a more detailed explanation presented in **Chapter 4: Implementation**. The system is designed by breaking down its functionalities into the following core modules:

- **User Authentication Module:** This module handles user login and ensures only authorized users can access stored data.
- **Password Storage Module:** Responsible for securely storing passwords in the encrypted SQLite database, ensuring efficient and secure data handling.
- **Encryption/Decryption Module:** Implements AES-256 and PBKDF2 encryption methods to protect passwords during storage and retrieval, ensuring strong data security.
- **Password Generator Module:** Allows users to generate strong, secure passwords based on customizable criteria, such as length, inclusion of alphabets, digits, and special characters, to enhance security.
- **Password Breach Check Module:** This module uses the Pwned Passwords API¹ to check if a password has been exposed in known data breaches. When a password is entered, the system hashes it using SHA-1 and queries the API to verify if it has appeared in any data breach. If the password has been compromised, users are alerted and advised to change their password, helping them avoid using vulnerable passwords and enhancing account security.

- **Password Strength Tester Module:** This module evaluates the strength of the password by simulating various attack methods, such as brute-force attacks, AES brute-force, dictionary attacks, and rainbow table attacks. The system estimates the time it would take to crack the password under each attack method and provides real-time feedback on the password's strength, allowing users to assess their password's resilience and make improvements as necessary.
- **AES Bit Evaluation Module:** This module evaluates the performance of different AES bit sizes (128, 192, and 256). The system measures the time taken to encrypt and decrypt passwords using various AES bit configurations and provides users with performance metrics such as encryption time, decryption time, memory usage, and throughput. This evaluation helps users understand the trade-off between encryption security and system performance, allowing them to choose the most suitable AES bit size for their needs.

1.5.4 Testing

The testing phase's objective is to verify that the system works as expected and meets all requirements. The focus will be on testing both the functionality, performance, and security of the system. Functional testing ensures that all features, such as password storage, generation, and password breach checks, work properly. Functional testing processes include unit testing for individual components, such as password hashing, and user acceptance testing (UAT) to assess the overall user experience and ensure the system meets user expectations. Performance testing evaluates how well the system performs under various conditions, such as stress, endurance, and scalability, ensuring that it can handle high loads and maintain efficiency. Security testing verifies that encryption methods, including AES-256 and PBKDF2, are applied correctly, ensuring that user data is securely stored and protected against potential threats.

1.5.5 Deployment

In the deployment phase, the objective is to prepare and distribute the application for end-users. The application will be compiled into an executable file (.exe) for use on Windows platforms, eliminating the need for installation. Users can run the system directly from the executable file without requiring any setup process. User documentation will be provided to assist with operating the software and understanding its features. The deployment process includes packaging the software, ensuring compatibility across various Windows versions, and conducting final checks for any potential issues before release.

1.5.6 Maintenance

The objective of the maintenance phase is to provide ongoing support and improvements to the system. After deployment, feedback from users will be collected to address any bugs or security vulnerabilities. The system will also receive updates as needed to improve its functionality and security features. Processes include monitoring for bugs, issuing patches, and making adjustments to ensure that the system remains efficient and secure over time.

1.6 Scope

Table 1.1 Project Scope

Section	Details
1. Project Deliverables	This project is a small-scale prototype designed to address fundamental password management needs. It targets individuals from various backgrounds, such as students, employees, unemployed individuals, and retirees, anyone needing a secure and convenient system for managing passwords. The prototype will focus on key features like password generation, password breach check, password strength tester, multi-factor authentication, and secure storage to provide a practical solution for password management.

	The project will deliver a functional prototype offering essential password management capabilities, excluding advanced features like cloud syncing, mobile applications, biometric authentication, browser integration, and auto-fill functionalities. By prioritizing critical features for secure password management, the prototype will focus on ease of use and efficient functionality for users. The project will use a single SQLite database to store password entries, including fields like platform, username, encrypted password, URL, notes, date added, and date modified. The dataset will be updated as users add, modify, or delete password records. All data will be stored locally, ensuring users can manage passwords without needing an internet connection.
2. In-Scope Features	Core features: - Master password setup with PBKDF2 hashing - Multi-factor authentication for an added security layer - Recovery key for access recovery - Password storage and retrieval - Password strength checker - Password strength tester - Password generation tool (secure, unique passwords) - Password breach checker - AES bit evaluation - AES-256 encryption for all stored passwords - Password categories and search functionalities Modules: - User authentication module - Password storage module - Encryption/Decryption module - Password generator module - Password breach check module - Password strength tester module - AES bit evaluation module Integration: - No external integrations (offline system)
3. Technology and Tools	Programming Languages: - Python (Python Software Foundation, 2001) - SQL (SQLite, n.d.)

	Platforms/Operating Systems: - Windows (desktop application) (Microsoft, n.d.) Development Tools: - Integrated Development Environment (IDE): Visual Studio Code (Visual Studio Code, n.d.) - Database: SQLite (SQLite, n.d.)
4. Users and Roles	Target Users: - General users (individuals seeking secure password management) Roles: - Can store, retrieve, and generate passwords
5. Constraints and Assumptions	Constraints: - Offline system; no internet or cloud functionality - No budget - Software must be developed before the end of 2nd semester session 2024/2025 Assumptions: - Users have basic computer knowledge - Software is used in offline environments - Windows will remain the primary platform for deployment
6. Out-of-Scope Features	Excluded Features: - Cloud-based storage or synchronization of passwords - Mobile app development - Biometric authentication features (e.g., fingerprint or facial recognition) - Browser integration - Auto-fill functionalities

Note. This table summarizes the project scope, including key deliverables, features, technologies, user roles, constraints, and exclusions. The project develops a secure offline password manager with essential features like password generation, encryption, and multi-factor authentication, excluding features like cloud storage and browser integration.

1.7 Significance of Project

This project is significant because it addresses the growing need for secure password management in a way that's accessible to everyone. With many online accounts, users often struggle to create strong passwords, leading to security risks. The system offers a solution with a password generator for creating strong, unique passwords, and a password breach check feature that alerts users if their passwords have been exposed in data breaches. The password strength tester simulates various attack methods, providing real-time feedback on password resilience and offering guidance for improvement.

Additionally, the system enables secure recovery of the password vault using a one-time recovery key, ensuring users can regain access if the master password is forgotten or compromised. It also uses strong encryption methods, including AES-256 and PBKDF2, to protect passwords from cyber threats. AES-256, a widely recognized encryption standard, uses a 256-bit key, making unauthorized access highly difficult (Menezes & Stebila, 2021). PBKDF2 enhances security by applying multiple cryptographic hash iterations, slowing down brute-force attacks (Choi & Seo, 2021). These methods together provide robust protection against unauthorized access and data breaches, ensuring confidentiality.

By combining these features—password generation, breach checking, strength testing, secure recovery, and strong encryption—this system provides a comprehensive and secure solution for managing passwords, enhancing user protection against digital threats.

1.8 Project Schedule

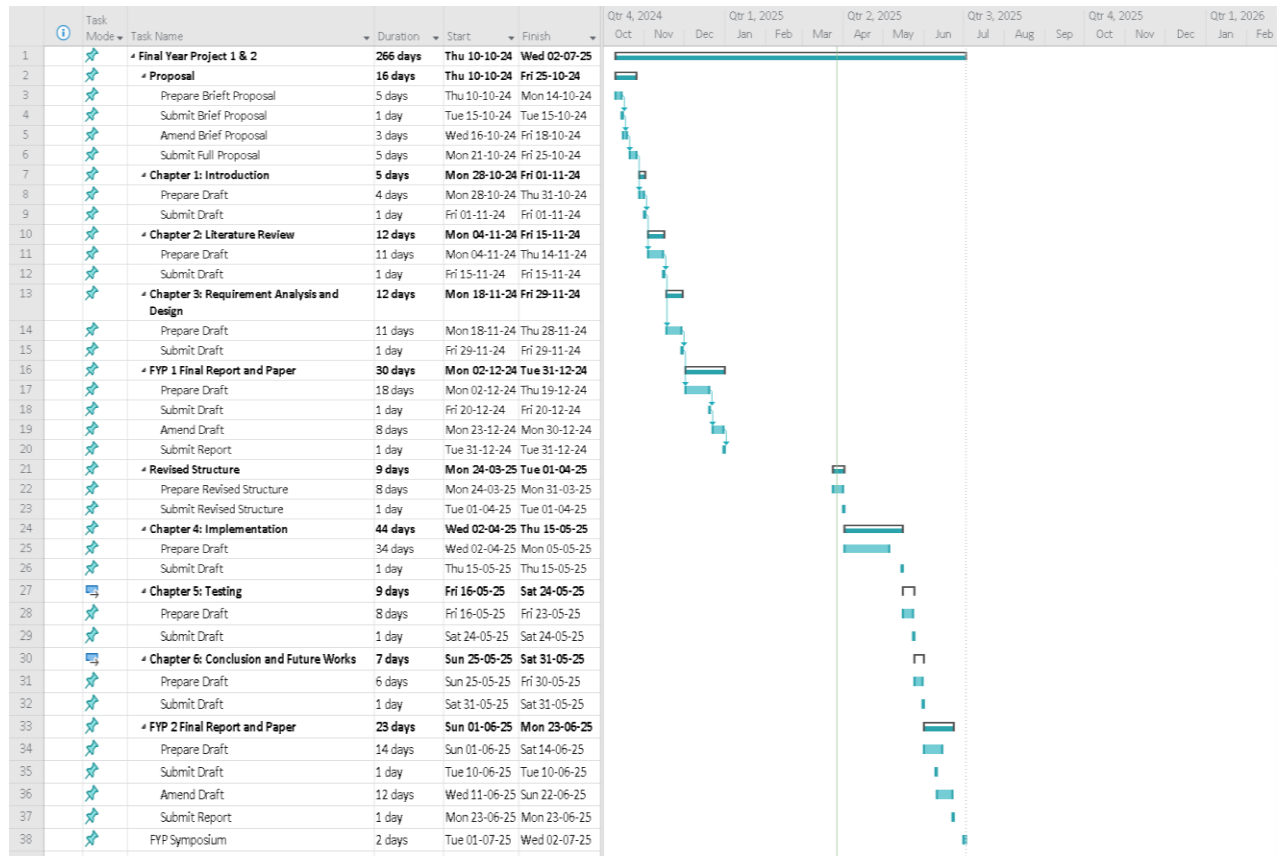


Figure 1.2 Gantt Chart for FYP 1 and FYP 2

Note. The Gantt chart provides a structured timeline for FYP 1 and FYP 2, outlining key tasks, durations, and dependencies. It visually represents the project's progress, including the proposal, report writing, and submission deadlines.

Figure 1.2 presents the Gantt Chart for Final Year Project (FYP) 1 and FYP 2, which outlines the planned timeline and sequence of tasks involved throughout the semester. This chart provides a visual overview of key phases such as topic selection, proposal writing, system planning, progress reporting, and final documentation. Each task is scheduled with defined start and end dates, along with dependencies that highlight the relationship between phases. The Gantt chart plays a crucial role in managing the project systematically by ensuring that each milestone is completed within the allocated timeframe. It also serves as a tracking tool to

monitor progress and make adjustments when necessary, thereby supporting effective time management and ensuring timely submission of all required deliverables.

1.9 Expected Outcome

The outcome of this project is a password management system, designed to help users easily manage and store their passwords securely through industry-level encryptions (AES-256 and PBKDF2). The system features a simple and minimalist design that makes it easy for anyone to navigate and use, allowing even those without technical skills to manage their passwords with minimal effort. This gives users peace of mind as they safely create accounts on various websites and platforms.

1.10 Summary

This chapter highlights the need for a secure password manager that addresses common issues such as weak, reused, and breached passwords, the risks of a single point of failure, and the importance of transparent password strength evaluation. The project follows the Waterfall Model to guide development through structured stages, ensuring a reliable and well-tested system that effectively meets its objectives: improving password security, minimizing vulnerabilities, and ensuring accessibility. Key features include AES-256 and PBKDF2 encryption, multi-factor authentication, a built-in password generator, password breach check, and password strength tester, all aimed at making password management secure and efficient. The chapter also outlines the scope, significance, schedule, and expected outcomes, providing a clear roadmap for developing a solution that enables users to securely store their passwords.

CHAPTER 2: LITERATURE REVIEW

2.1 Overview

This chapter provides an overview of encryption and its role in secure password management, focusing on AES and PBKDF2. It also investigates existing password management systems, including 1Password, Keeper, Total Password, and Sticky Password. The chapter begins with an introduction to encryption, AES, PBKDF2 and how they work together, followed by an analysis of the features, advantages, and disadvantages of each system. A comparative analysis will be presented in a table, along with a detailed description. Finally, we will identify key features from these systems that could be incorporated into the proposed password management system prototype.

2.2 Encryption Methods: AES and PBKDF2

This section explains the concept of encryption, followed by an overview of AES and PBKDF2, and concludes with how these two encryption methods collaborate to ensure robust security for password storage and management.

2.2.1 Encryption

Encryption is the process of converting data into a coded format to prevent unauthorized access. This is crucial in securing sensitive information, such as passwords, by transforming readable data into an unreadable form, which can only be reversed with the appropriate decryption key (Archana & Niranjana, 2023). Encryption plays a pivotal role in protecting stored and transmitted data, making it an essential part of any secure system. By ensuring that only authorized parties with the correct decryption key can access the original data, encryption

helps maintain the confidentiality, integrity, and authenticity of sensitive information (Liu, 2024).

2.2.2 AES (Advanced Encryption Standard) Encryption

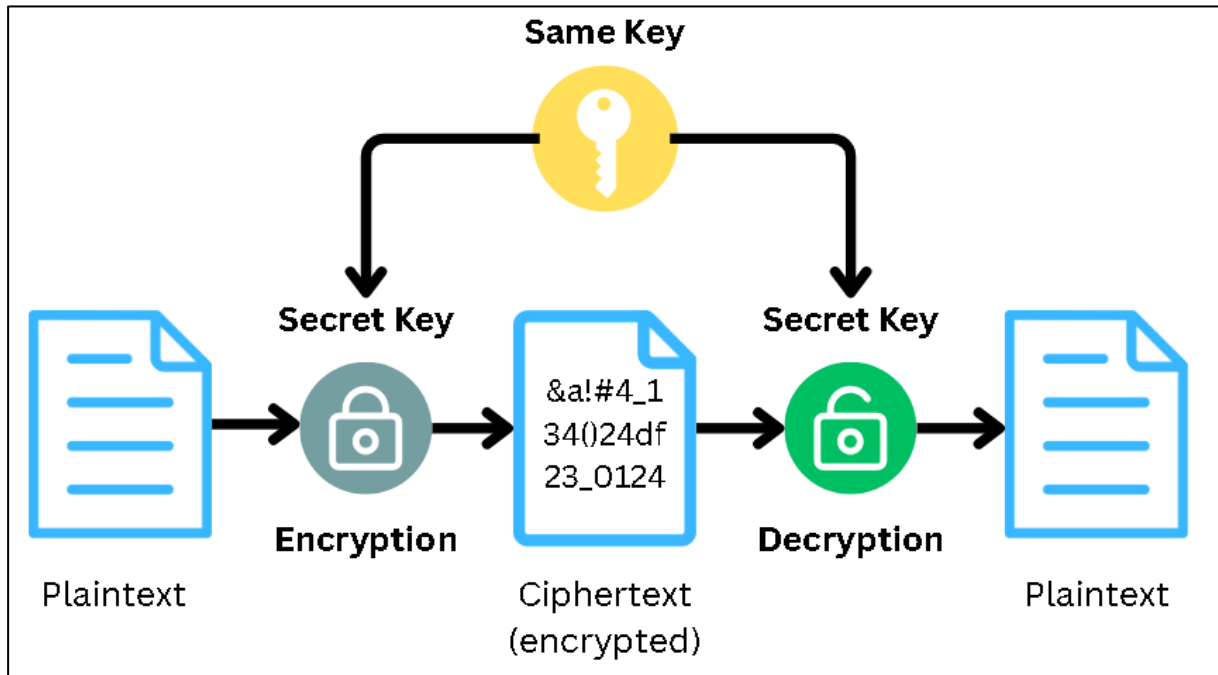


Figure 2.1 AES Encryption and Decryption Process

Note. This figure illustrates the AES encryption and decryption process, where the same key is used for both converting plaintext into ciphertext and reversing the ciphertext back to plaintext.

AES is a widely used symmetric encryption algorithm known for its strong security, efficiency, and versatility. It operates on 128-bit data blocks and supports key sizes of 128, 192, or 256 bits, with security increasing as key length grows. Figure 2.1 illustrates the encryption and decryption process, where the same key is used for both operations, converting plaintext to ciphertext and vice versa. The encryption process involves several rounds—SubBytes, ShiftRows, MixColumns, and AddRoundKey—which protect against cryptographic attacks like brute force and differential cryptanalysis (Sarkar et al., 2024). AES is favored for its speed and low resource usage, making it ideal for applications in mobile devices, cloud storage, and

IoT. While AES is highly secure, proper implementation is crucial to avoid vulnerabilities, and recent studies explore enhancements to improve its security and performance, albeit with some added computational cost (Abikoye et al., 2019). Despite its error propagation characteristics, AES remains a trusted standard for securing digital communications and data storage.

2.2.3 PBKDF2 (Password-Based Key Derivation Function 2)

PBKDF2 is an essential technique used in password management systems to securely derive a key for the chosen encryption algorithm. It works by applying a cryptographic hash function, typically HMAC with SHA-1 or SHA-256, to the password and a dynamic salt, which is a unique random value generated for each password. The process involves multiple rounds of hashing, significantly increasing the computational effort required to perform brute-force or dictionary attacks (Choi & Seo, 2021). The dynamic salt ensures that identical passwords result in different hashes, preventing the use of precomputed hash databases (rainbow tables). The number of iterations and the strength of the salt determine the effectiveness of PBKDF2; more iterations make the function slower but more resistant to attacks. The resulting derived key is used for encrypting the password or other sensitive data, offering robust protection against common attacks, especially when implemented with a dynamic salt in password management systems.

2.2.4 Integration of AES and PBKDF2 in Password Encryption

In the proposed password management system, AES and work together to provide a high level of security for stored passwords. Since user-chosen passwords are typically weak and unsuitable as cryptographic keys, PBKDF2 is employed to transform the password into a robust, cryptographically secure key (Ali et al., 2021). This is achieved by applying multiple rounds of hashing with a dynamic salt, making it computationally expensive for attackers to

attempt brute-force or dictionary attacks. Once the key is derived, AES uses it to securely encrypt the password, ensuring that sensitive user data remains protected from unauthorized access (Sarkar et al., 2024). The combination of AES encryption and PBKDF2 key derivation offers strong protection against common password-based attacks, making the system highly secure, even for weaker passwords. This dual-layer approach ensures that the system effectively defends against various attacks, including brute-force, dictionary, and rainbow table attacks, offering users a robust and secure password management solution.

2.3 Review on Similar Existing System

This section examines the overviews, features, advantages, and disadvantages of the existing systems. Four similar systems have been selected as references for this project: 1Password, Keeper, Total Password, and Sticky Password.

2.3.1 1Password

1Password² is a password management solution developed by AgileBits Inc., designed for individuals, families, and businesses to store, generate, and manage passwords and sensitive data (AgileBits, n.d.). The system is renowned for its password organization, alerts for password hygiene, and credential capture and replay features, which contribute to its reputation in enhancing digital security for both personal and professional use. It also offers cross-platform compatibility, supporting major operating systems such as Windows, macOS, Linux, Android, and iOS. As of 2024, 1Password serves over 15 million users worldwide and accounts for approximately 7% of the total password management market share (Team SignHouse, 2024).

² 1Password. *AgileBits Inc.* Available at: <https://1password.com/>

Figure 2.2 shows the main menu of 1Password, while Figure 2.3 displays the add item options available in 1Password, including login, credit card, notes, and other items.

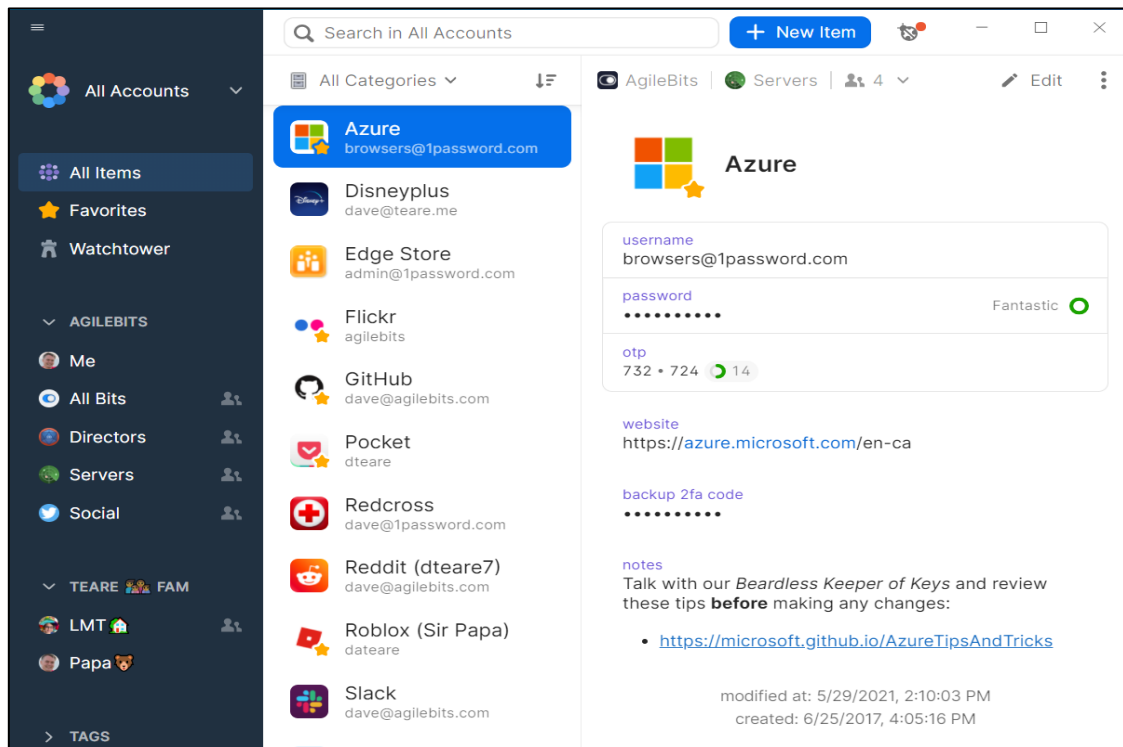


Figure 2.2 Main Menu for 1Password

Note. The main menu of 1Password provides an organized interface for managing stored credentials securely. The left panel categorizes stored accounts, while the central panel displays a selected entry's details, including the username, password, OTP, website, and additional notes. This layout ensures easy access and management of sensitive information while maintaining security and efficiency.

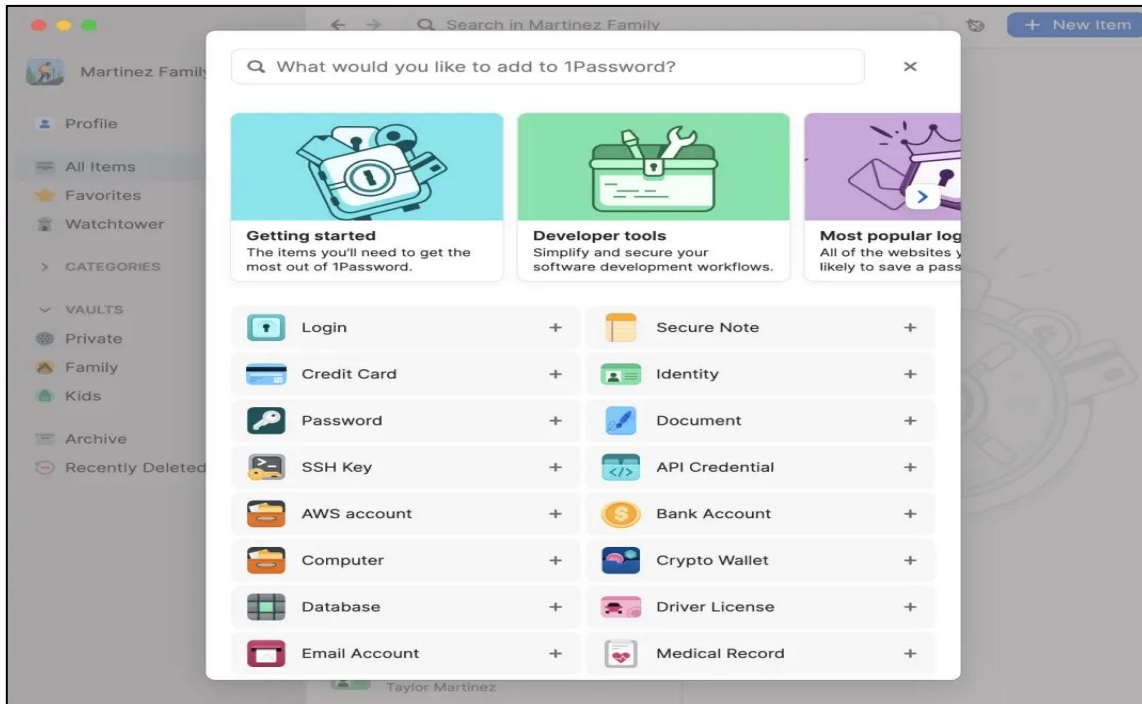


Figure 2.3 Add Item Options in 1Password

Note. The "Add Item" menu in 1Password provides users with a variety of options to securely store different types of sensitive information. Categories include login credentials, credit card details, SSH keys, API credentials, bank accounts, medical records, and more. This feature helps users organize and manage their important data efficiently within a secure vault.

1Password offers a range of key features designed to enhance both security and convenience for users. One standout feature is travel mode, which allows users to temporarily remove sensitive information from their devices while traveling. This ensures that if a device is lost or stolen, critical data such as passwords, credit card details, or other private information remains protected. Travel Mode can be easily activated or deactivated as needed.

Another important feature is password health checker, which helps users maintain strong and secure passwords. It analyses stored passwords and offers feedback on their strength, suggesting improvements when necessary. In addition, Watchtower, 1Password's data breach alert system, constantly monitors for breaches and alerts users if their stored passwords are exposed in any known breaches. Figure 2.4 shows the dashboard for Watchtower and the

password health checker. Together, these tools support digital security by promoting best practices and providing proactive alerts to safeguard sensitive information.

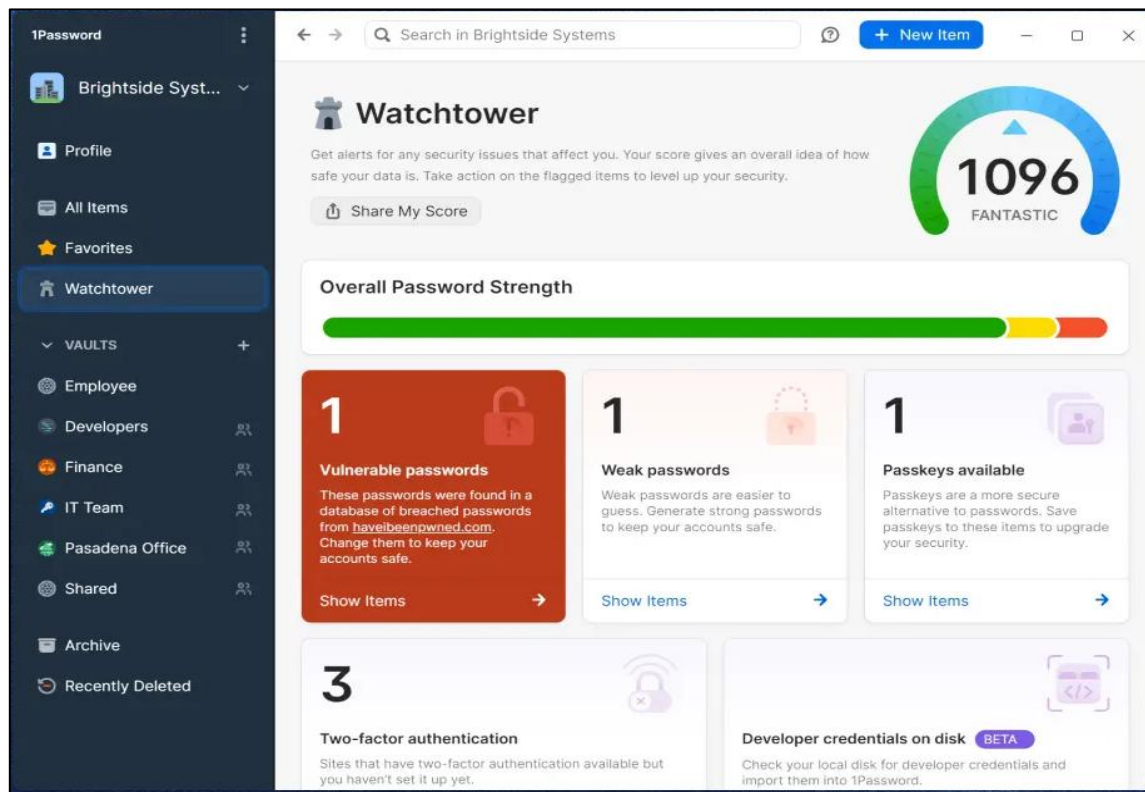


Figure 2.4 Dashboard for Watchtower and Password Health Checker

Note. The Watchtower dashboard in 1Password offers a detailed overview of password health and security alerts, highlighting vulnerabilities and weak passwords. It also encourages users to enable two-factor authentication and provides tools to enhance overall account safety.

1Password offers several key advantages that make it a popular choice for many users. First, it allows users to access their stored passwords without an internet connection. This feature is helpful when there is no internet available, ensuring users can still get their passwords when needed. Besides that, 1Password uses AES-GCM-256 encryption, a symmetric key algorithm with a 256-bit key that secures data by combining AES and GCM for both encryption and authentication (Preez, 2024). This method is widely used to protect sensitive information

from unauthorized access. Moreover, 1Password has recovery keys for account recovery. If users forget their master password, they can use these recovery keys to regain access to their accounts. These advantages make 1Password a useful and secure tool for managing passwords.

However, there is a notable drawback to consider. 1Password does not offer a portable version, meaning users cannot run the application directly from a USB drive without installing it on each device. A portable version would enable users to access their passwords on multiple computers without needing to install the software on each one (Soft Perfect, 2024). This feature would be especially helpful for users who often work on different devices or need to access their passwords while traveling. Without this option, users must install 1Password on every device they wish to use, which may not be convenient for everyone. While this limitation might not be a concern for all users, it is an important factor to keep in mind when deciding if 1Password is the right choice.

2.3.2 Keeper

Keeper Password Manager³ is developed by Keeper Security, Inc., a company founded in 2009 by Craig Lurey and Darren Guccione (Keeper Security, n.d.). Headquartered in Chicago, Illinois, Keeper Security has grown significantly. By 2024, more than 46 companies worldwide were using Keeper Password Manager for password management, with the majority based in the United States, making up 76.09% of its user base. Other countries using Keeper include Germany and Canada, each contributing a smaller portion of users (6sense, 2024). Keeper offers various plans tailored for individuals, families, and businesses, along with a free trial, providing

³ Keeper Password Manager. *Keeper Security, Inc.* Available at: <https://www.keepersecurity.com/>

a convenient solution for managing logins and sensitive data. Figure 2.5 shows the main menu of Keeper, while Figure 2.6 illustrates the item-adding feature.

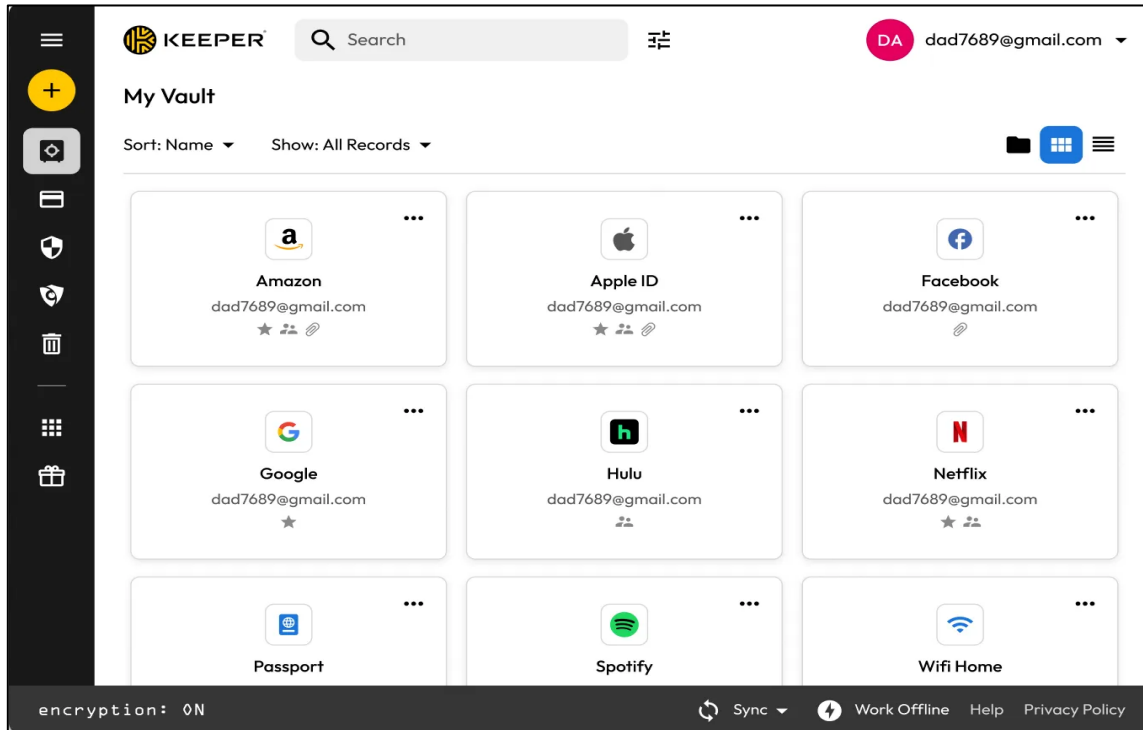


Figure 2.5 Main Menu for Keeper

Note. The main menu for Keeper displays a user's vault with various stored credentials such as Amazon, Apple ID, Facebook, and more. The interface includes options to sort and filter records, manage encryption, sync data, and access additional features like offline mode and help resources.

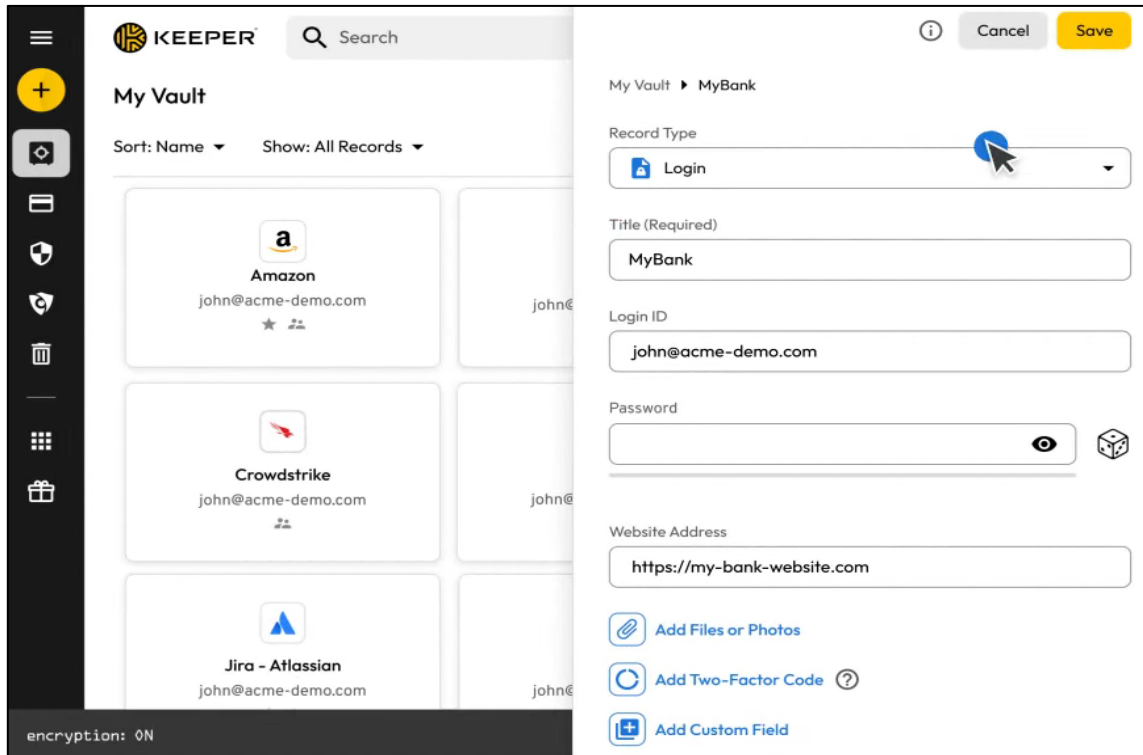


Figure 2.6 Add Item Option in Keeper

Note. The "Add Item" option in Keeper allows users to securely store and manage various types of credentials. Users can input details such as login ID, password, and website address, as well as add files, photos, two-factor codes, and custom fields for enhanced security and organization.

Keeper offers several features that help manage passwords and personal information. One of the key features is the auto-fill option for personal data. This tool automatically fills in online forms with information the user has saved, such as name, address, or credit card details. By clicking the "fill record" button, users can quickly complete forms. As shown in Figure 2.7, this feature simplifies the process of entering personal details on websites.

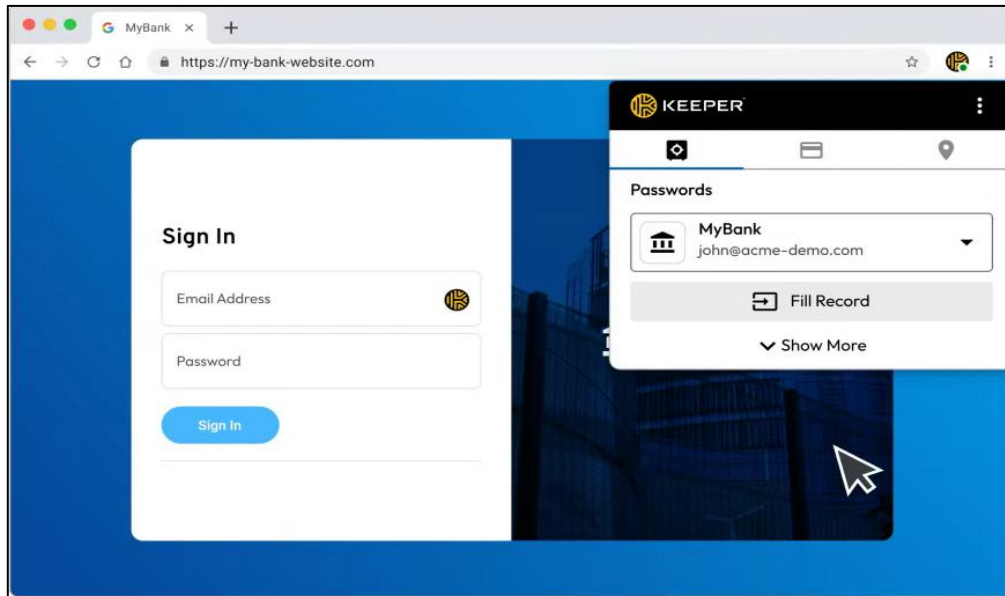


Figure 2.7 Auto-fill Feature in Keeper

Note. The auto-fill feature in Keeper simplifies the login process by automatically populating stored credentials such as email addresses and passwords. This feature enhances user convenience while maintaining security by ensuring that sensitive information is easily accessible yet protected.

In addition to auto-fill, Keeper uses AES-256 encryption, a strong encryption method designed to protect data. AES-256 encryption is a cryptographic algorithm that uses a 256-bit key to scramble plain text into ciphertext (Nurhandhi & Suhendar, 2023). With AES-256, users' passwords and personal details are secured using one of the most effective encryption methods available, ensuring that information remains private even if there is a data breach. This encryption makes Keeper a reliable choice for anyone looking to protect sensitive information.

Keeper brings several noteworthy benefits to the table. One significant advantage is its support for multi-factor authentication, which strengthens security by requiring additional verification methods beyond just a password. This added layer of protection makes it harder for unauthorized individuals to breach accounts, providing peace of mind for users who value their digital safety. Additionally, Keeper offers a backup and restore function, allowing users to recover their passwords and sensitive information if they encounter data loss or technical issues.

This feature ensures that users can maintain access to critical information without problems, providing a reliable safety net for those concerned about data integrity.

However, Keeper also has its limitations. One major limitation is the lack of a portable option, similar to 1Password, as mentioned above. Another limitation is the lack of a password health checker, which helps users check the strength and security of their passwords. Without this feature, users may overlook helpful information that could improve their security, leaving them open to cyber threats. These limitations should be considered when looking at Keeper as a password management tool.

2.3.3 Total Password

Total Password⁴ is developed by TotalAV, a company known for its comprehensive security solutions (Total Password, n.d.). Total Password is designed to manage users' logins and sensitive information securely. TotalAV, based in the United Kingdom, has expanded its offerings to include this password management tool as part of its broader security suite. The platform has gained popularity for its integration with TotalAV's Total Security plan, which includes additional security tools for added secure (Oguejiofor, 2024). Total Password is available at an introductory price of \$1.99 per month, with a 30-day money-back guarantee. Figure 2.8 shows the main menu of Total Password.

⁴ Total Password. *TotalAV*. Available at: <https://www.totalpassword.com/>

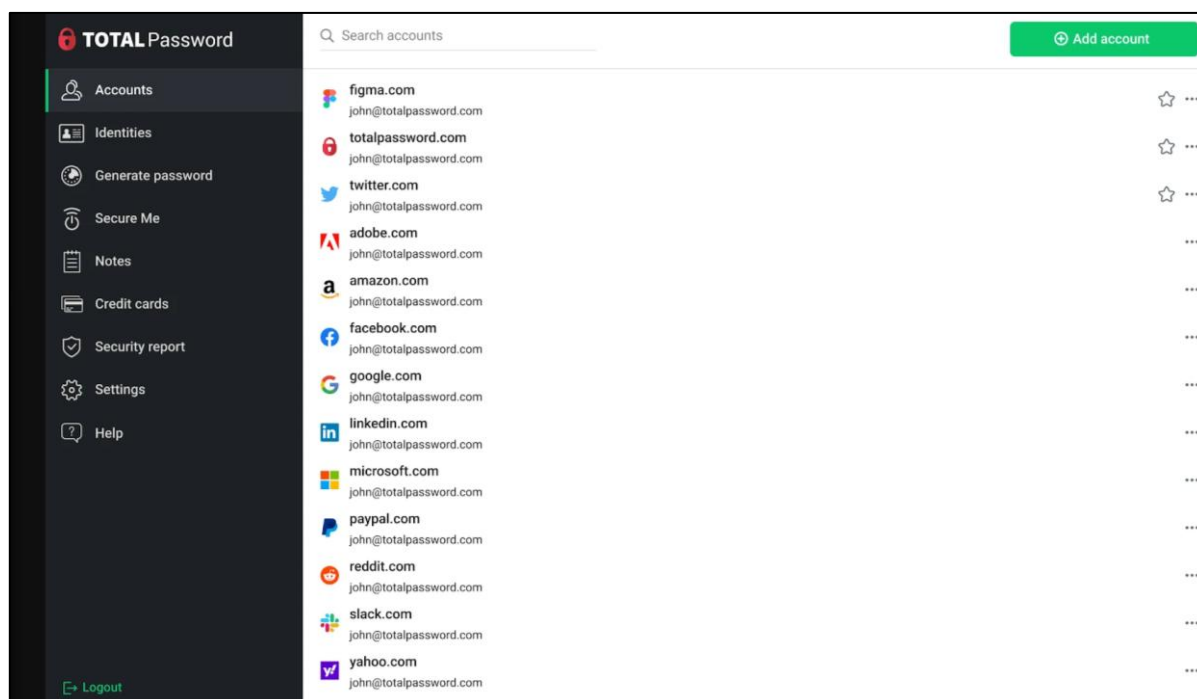


Figure 2.8 Main Menu for Total Password

Note. The main menu for Total Password provides a comprehensive interface for managing various accounts and credentials. It includes options to generate passwords, view security reports, and manage identities, notes, and credit cards. The search feature allows users to quickly locate specific accounts, enhancing both security and usability.

Total Password offers essential features designed to keep users' accounts secure and provide peace of mind. One of its key features is data breach detection, which continuously monitors stored passwords for any signs of exposure in known breaches. If one of the passwords is found in a breach, Total Password will alert users immediately, allowing them to take quick action to change compromised passwords and protect their accounts from potential threats.

Additionally, Total Password provides a security report, as shown in Figure 2.9, which gives users a detailed overview of their password health and security status. This report helps users evaluate the strength of their stored passwords and offers suggestions for improvement, identifying weak or reused passwords to ensure that users follow best practices for securing sensitive data.

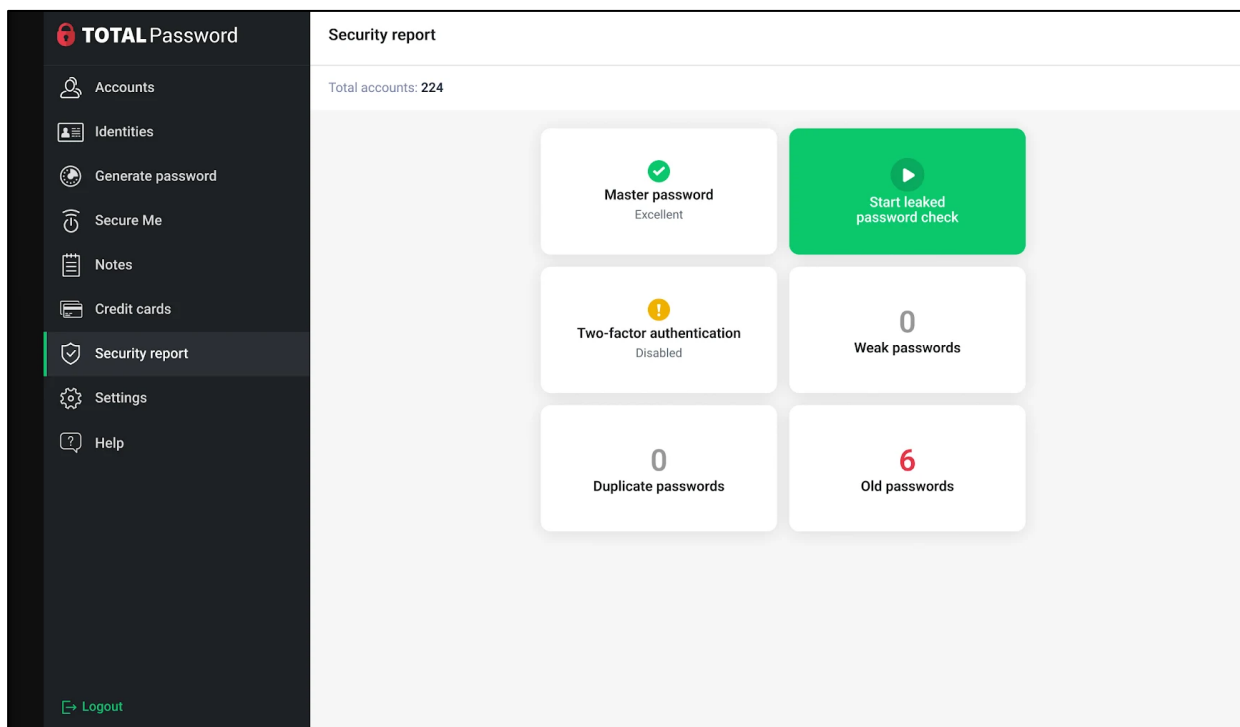


Figure 2.9 Security Report in Total Password

Note. The Security Report in Total Password provides a detailed analysis of the user's password health and security status. It highlights areas such as weak passwords, duplicate passwords, and old passwords, and offers options to start a leaked password check and enable two-factor authentication. This feature helps users identify and address potential security vulnerabilities to enhance their overall account safety.

Total Password offers several advantages that make it a useful option for password management. One advantage is its brute force protection feature, which safeguards against unauthorized access by limiting incorrect login attempts. This improves security by preventing hackers from using persistent guessing techniques to breach accounts. Moreover, Total Password's support for biometric login such as fingerprint or facial recognition adds an extra layer of security while making the authentication process faster and more convenient.

On the downside, there are some disadvantages to consider. It is only available through a paid subscription, which may not be suitable for users who want a free or trial version. It does not offer offline access, so users can only get their passwords with an internet connection. It also lacks an autofill feature for personal data, requiring users to enter information manually

into forms. The app also cannot be run from a USB drive, so it must be installed on each device. Lastly, Total Password does not support password sharing, which may be an issue for users who need to share passwords. These points should be considered when evaluating the tool.

2.3.4 Sticky Password

Sticky Password⁵ is developed by Lamantine Software, a company specializing in security solutions (Lamantine Software, n.d.). Founded in 2001 and based in the Czech Republic, Lamantine Software has focused on creating user-friendly and secure password management tools. Sticky Password is designed to manage users' logins and sensitive information securely. The platform has gained recognition for its robust security measures and unique features, such as dark web monitoring and a portable USB password manager (John, 2024). Sticky Password offers both free and premium plans, with the premium plan available at a competitive price and a 30-day money-back guarantee. Figure 2.10 shows the main menu of Sticky Password.

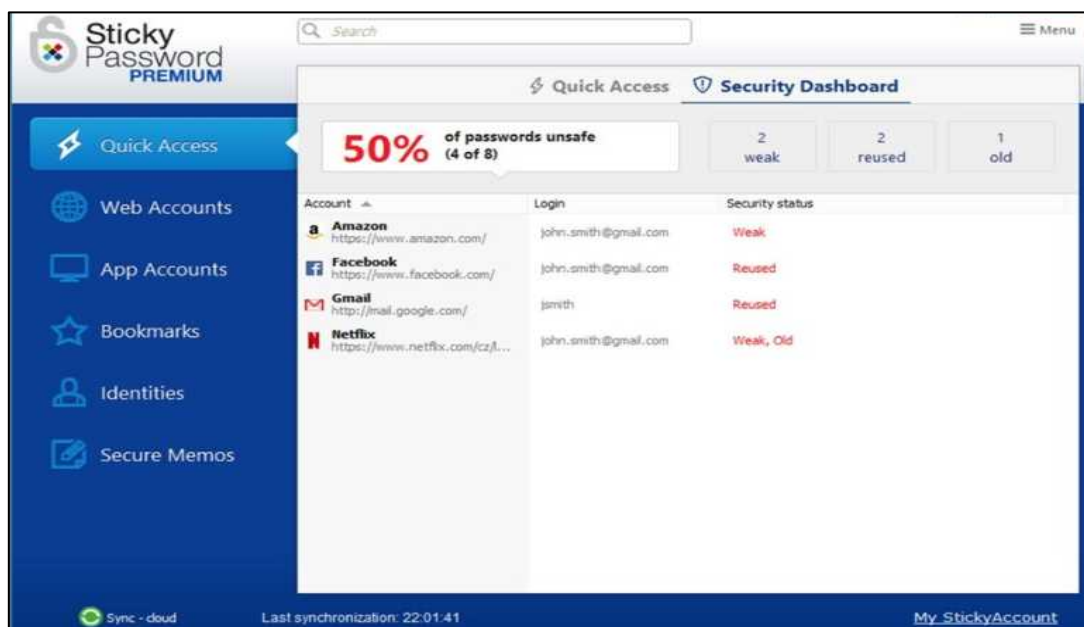


Figure 2.10 Main Menu for Total Password

⁵ Sticky Password. Lamantine Software. Available at: <https://www.stickypassword.com/>

Note. The main menu for Sticky Password provides a simple and easy-to-use interface for managing web and app accounts, identities, and secure memos. It highlights security statuses such as weak, reused, and old passwords, helping users identify and address potential vulnerabilities. The interface also provides quick access to synchronization status and premium features, ensuring efficient and secure password management.

Sticky Password includes features designed to provide both functionality and security. One key feature is the portable option, which lets users install the application on a USB drive. This allows users to run the application directly from the USB without installing it on individual computers, enabling password access on multiple computer devices while maintaining privacy and security. It also adds convenience for use in different locations.

Another valuable feature of Sticky Password is its password health checker, as illustrated in Figure 2.11, which helps users evaluate the strength of their passwords. The tool scans stored passwords and identifies any that are weak, reused, or at risk of being compromised. It then provides suggestions for improving password security, helping users ensure that their accounts are as safe as possible. This feature is a key part of Sticky Password's commitment to helping users maintain high security standards without the hassle of manual password checks.

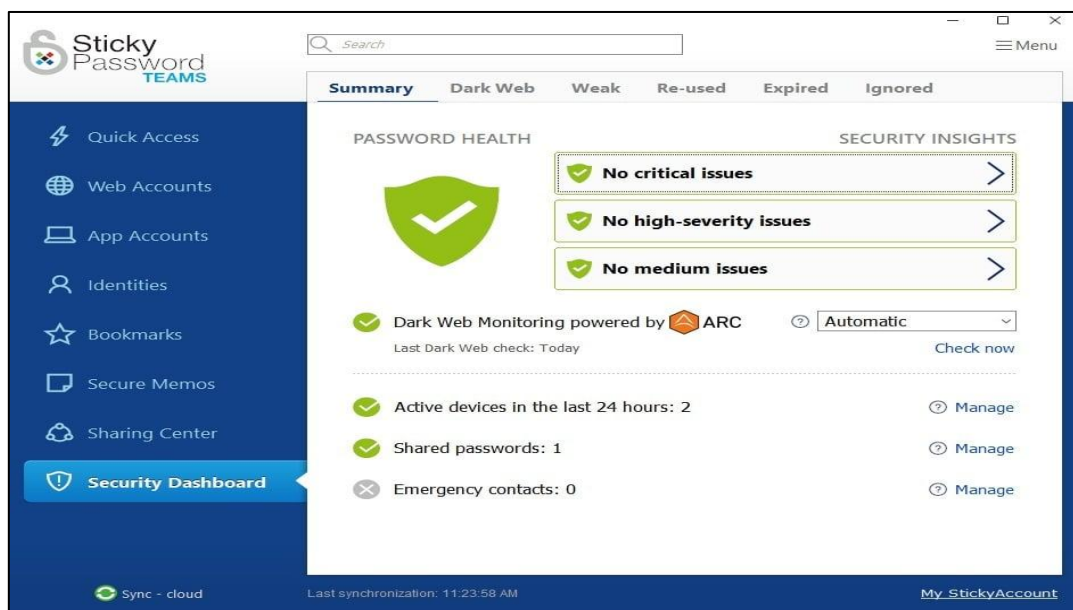


Figure 2.11 Password Health Checker for Stick Password

Note. The Password Health Checker in Sticky Password provides a comprehensive overview of password security, highlighting issues such as weak, re-used, and expired passwords. It includes features like Dark Web Monitoring and a Security Dashboard to help users manage and improve their password health.

Sticky Password offers several advantages that enhance both convenience and security. One major advantage is its password health checker, which allows users to assess the strength of their passwords and receive suggestions for improvement. Another advantage is the auto-fill function for personal data, which simplifies the process of filling out online forms, saving users time. Additionally, Sticky Password has the advantage of portability, unlike 1Password, Keeper, and Total Password. Users can store their password manager on a USB drive, allowing them to securely access their passwords on any device without needing to install the software. This portability ensures users can maintain security and convenience wherever they go.

However, there are some downsides to consider. One significant drawback is that Sticky Password does not offer offline access, meaning users can only access their passwords when connected to the internet. This can be a big inconvenience if users are in a place with no internet connection. Another major disadvantage is the lack of a recovery method if users forget their master password. The only option is to reset the account, which means losing all stored data. This could be a serious issue for users who rely heavily on their password manager for access to important accounts and information. These factors are important to keep in mind when deciding if Sticky Password is the right choice for users.

2.4 Comparison of Features between Existing Systems and Proposed System

In this section, the systems will be compared based on their features, as shown in Table 2.1. This table will summarize the comparison between the existing systems and the proposed prototype, providing a comprehensive overview.

Table 2.1 Comparison Table of Existing System and Proposed Prototype

System Feature	1Password	Keeper	Total Password	Sticky Password	Proposed Prototype
Cost	Paid (Free Trial 14 days)	Paid (Free trial 30 days)	Paid (No free trial)	Paid (Free trial 30 days)	Free
Offline Access	Yes	Yes	No	No	Yes
Store Password in Cloud	Yes	Yes	Yes	Yes	Out of Scope
Password Health Checker	Yes	No	Yes	Yes	Yes
Password Generator	Yes	Yes	Yes	Yes	Yes
Password Sharing	Yes	Yes	No	Yes	Out of Scope
Multi-factor Authentication	Yes	Yes	Yes	Yes	Yes
Brute Force Protection	Yes	Yes	Yes	Yes	Yes
Backup and Restore	Yes	Yes	Yes	Yes	Yes
Real-time Alerts	Yes	Yes	Yes	Yes	Yes
Travel Mode	Yes	No	No	No	Out of Scope
Data Breach Detection	Yes	Yes	Yes	Yes	Yes
Biometric Login	Yes	Yes	Yes	Yes	Out of Scope
Auto-Fill Password	Yes	Yes	Yes	Yes	Out of Scope
Auto-Fill Personal Data	Yes	Yes	No	Yes	Out of Scope

Security Report	Yes	Yes	Yes	Yes	Yes
Portable	No	No	No	Yes	Yes
Encryption Type	AES-GCM-256	AES-256	AES-256	AES-256	AES-GCM-256, with PBKDF2
Recovery Method if Master Password is Forgotten	Recovery keys	Email or phone support	Email	Not available	Recovery keys

Note. This table compares existing password managers with the proposed prototype based on key features like access, cost, and functionality. The prototype offers a free solution with essential features such as password health checks, sharing, and strong encryption while excluding cloud storage and travel mode. Master password recovery uses recovery keys for added security.

From the comparison in Table 2.1, all the selected password managers include features such as password generators, multi-factor authentication, and autofill password functionality, which are essential for a password manager (Baskar et al., 2024). These features help users create strong passwords, eliminate the need to memorize or manually record passwords, and enhance security while making them more convenient to use (Taneski et al., 2019). All of them also have backup and restore options, allowing users to recover their passwords if issues arise. This ensures protection against data loss from device theft, accidental deletions, or other unexpected situations (Master, 2022).

However, some password managers have unique features that make them different. For example, 1Password offers a travel mode that temporarily removes sensitive data from a user's device, providing added security while traveling and reducing the risk of exposure in case of device theft or inspection. Additionally, it includes recovery keys, which allow users to regain access to their passwords if they forget their master password. Sticky Password offers portable options, enabling users to use the password manager on a USB drive for quick access on any computer without needing to install the software.

Some features are missing in certain password managers, which might affect how users use them. For instance, Total Password and Sticky Password lack offline functionality, which require users need to have an active internet connection to access their passwords. Total Password also lacks password sharing, meaning users cannot share their passwords with others easily, which could be a problem for those who need to collaborate or grant access to accounts. Besides that, Keeper lack a password health checker, which means users cannot assess the strength of their passwords within the system. This absence could lead to weaker or vulnerable passwords being used, potentially compromising security. Furthermore, Sticky Password does not have a way to recover an account if the master password is forgotten, so users would have to reset their account and lose all stored data. These differences are important to consider when choosing the best password manager for individual needs.

2.5 Brief Overview of Proposed System

When reviewing existing password management systems, it becomes clear that a new prototype should address key limitations. The proposed system will provide essential features such as offline access, AES-256 encryption, salted hashing, PBKDF2 SHA-256 and password health checker. Unlike many current solutions, it will not rely on cloud storage, instead focusing on local data management to reduce the risks associated with storing passwords on external servers (Ayyagari et al., 2019).

Moreover, the prototype will incorporate essential features such as brute force protection, data breach detection, real-time alerts, backup and restore functions, multi-factor authentication, and recovery keys for retrieving the master password, providing users with greater control over their accounts and ensuring a higher level of protection against

unauthorized access. Additional features include password generation, brute force protection, and password health checks. USB portability will enable users to securely store and access passwords on different computers easily. These features ensure users retain control over sensitive data while providing robust protection against unauthorized access.

The system does not include features such as password sharing, auto-fill for passwords and personal information, travel mode, or biometric login because these functionalities are outside the scope of the current project. The primary focus is to develop a working offline password manager with privacy controls as the foundation. Implementing these additional features would require more time, tools, and resources than planned, which could shift focus from the main objectives and cause delays. However, after the system is launched, becomes stable, and is well-received, these features will be added in future updates. This approach allows the system to address basic user needs first, ensuring an on-time launch and providing immediate value. By concentrating on core features, the system remains reliable and easy to use while planning for future improvements. Table 2.1 outlines how this plan addresses gaps in existing solutions and sets a path for future updates.

2.6 Review of Tools and Technology Used

Programming Languages:

- **Python:** Python⁶ is chosen for this project due to its simplicity, readability, and versatility. It is well-suited for building desktop applications and provides extensive libraries for tasks like encryption, user authentication, and file management. Python's

⁶ Python. *Python Software Foundation*. Available at: <https://www.python.org/about/>

robust ecosystem allows for the efficient development of password management functionality, including integrating AES-256 encryption and multi-factor authentication.

- **SQL:** SQL⁷ is used to manage and query the SQLite database, which stores all password-related data. SQL allows for efficient data retrieval and manipulation, ensuring secure storage and retrieval of passwords. It is essential for maintaining the database's structure, which includes fields like platform, username, encrypted password, and other password-related details.

Platforms/Operating Systems:

- **Windows 11 (Desktop Application):** The project is developed as a desktop application for Windows 11⁸, a widely used operating system. Windows 11 offers a stable environment for running applications, and building the application for this platform ensures compatibility with a broad range of user devices. The use of a desktop platform also facilitates offline functionality, which is a core requirement of the project.

Development Tools:

- **Integrated Development Environment (IDE):** Visual Studio Code (VS Code)⁹ is used as the IDE for this project due to its lightweight nature and support for Python. It offers features like syntax highlighting, debugging, and version control integration, which streamline the development process. VS Code's versatility and extension capabilities make it ideal for building and testing Python applications.

⁷ SQL Introduction. *W3Schools*. Available at: https://www.w3schools.com/sql/sql_intro.asp

⁸ Windows 11. *Microsoft*. Available at: <https://www.microsoft.com/en-us/windows?r=1>

⁹ Visual Studio Code. *Microsoft*. Available at: <https://code.visualstudio.com/>

- **Database:** SQLite¹⁰ is a serverless, self-contained database engine that is embedded within the application. It is used for this project because it is lightweight, easy to set up, and operates locally without the need for an external server. This aligns with the project's offline functionality, allowing password data to be stored securely on the user's device without requiring an internet connection.

2.7 Summary

In summary, an analysis of different password management systems has been conducted, examining their key features, advantages, and disadvantages. Based on the analysis, decisions were made regarding the features to include in the proposed password management system prototype. To achieve the project's goals within the constraints of time, budget, and available knowledge, only a limited set of features will be implemented. Future recommendations include enhancing compatibility with additional operating systems, developing a web-based version, incorporating data breach detection, enabling biometric authentication, and introducing auto-fill functionality for forms and passwords.

¹⁰ SQLite. *SQLite.org*. Available at: <https://www.sqlite.org/>

CHAPTER 3: REQUIREMENT ANALYSIS & DESIGN

3.1 Overview

This chapter employs the Waterfall methodology to develop the password management system prototype, starting with the Requirement Analysis phase, where user data is gathered through a survey to identify key features for the prototype. This is followed by the System Design phase, which employs Use Case Diagram, Sequence Diagrams, and Class Diagram to define the system's functional requirements and illustrate user interactions. A flowchart is also created to demonstrate the secure storage and retrieval of passwords using AES-256 encryption and PBKDF2 hashing. Additionally, the user interface design is detailed, showcasing the main menu, add/edit password forms, settings menu, and password health checker menu, which outline the system's layout and user navigation.

3.2 Requirement Analysis

The goal of this phase is to define the prototype's requirements by understanding user needs and ensuring the system meets those needs. This involves identifying user requirements and documenting detailed specifications that will guide the design and implementation phases.

3.2.1 Method for Gathering Requirements

For gathering the requirements of the proposed password management system, a survey was conducted using Google Forms. The survey included 20 respondents aged between 18 and 64 years old, representing diverse backgrounds such as students, employed individuals, self-employed people, unemployed individuals, and retirees. The respondents were randomly

selected to ensure varied perspectives on the requirements. The survey results, presented in Figure 3.1 to Figure 3.6, covered aspects such as the number of online accounts users manage, their password reuse habits, and their preferences for password management systems. The detailed questions from the survey can be found in Appendix A. The findings from this survey will inform the design and development of the system to address user needs effectively.

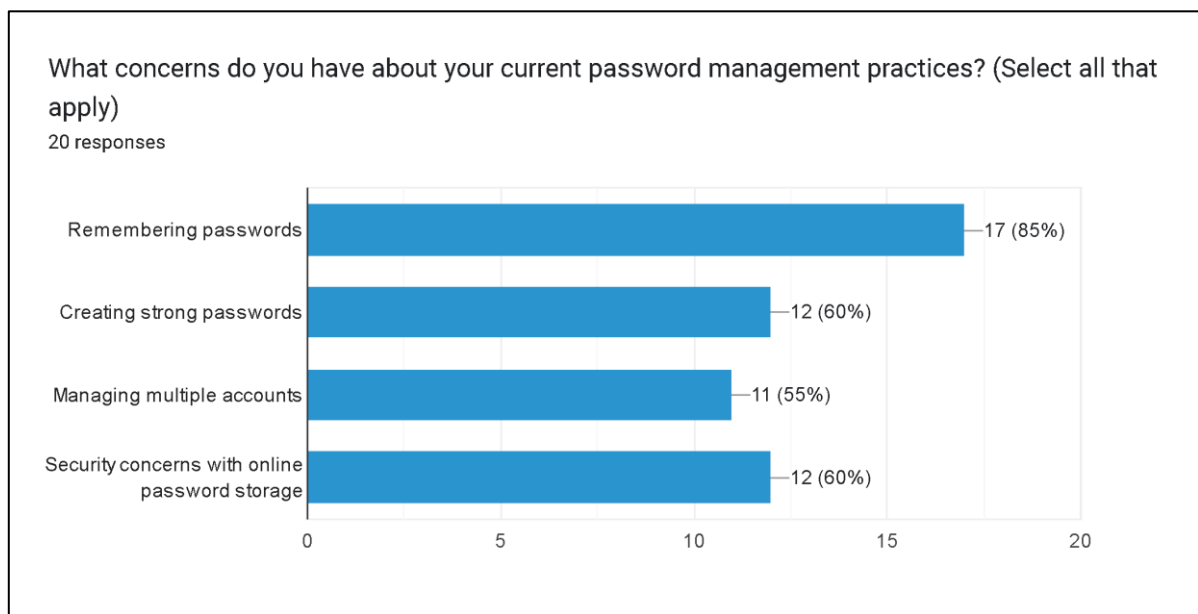


Figure 3.1 Concerns on Password Management Practices

Note. This figure highlights common concerns users have about their current password management practices. These findings underscore the challenges users face in maintaining secure and efficient password practices.

Figure 3.1 displays the concerns respondents have regarding their current password management practices. 85% of respondents are most concerned about remembering passwords, followed by 60% who are worried about creating strong passwords and the security of online password storage. 55% of respondents are concerned about managing multiple accounts. These results suggest that respondents are aware of the challenges involved in managing passwords securely and effectively.

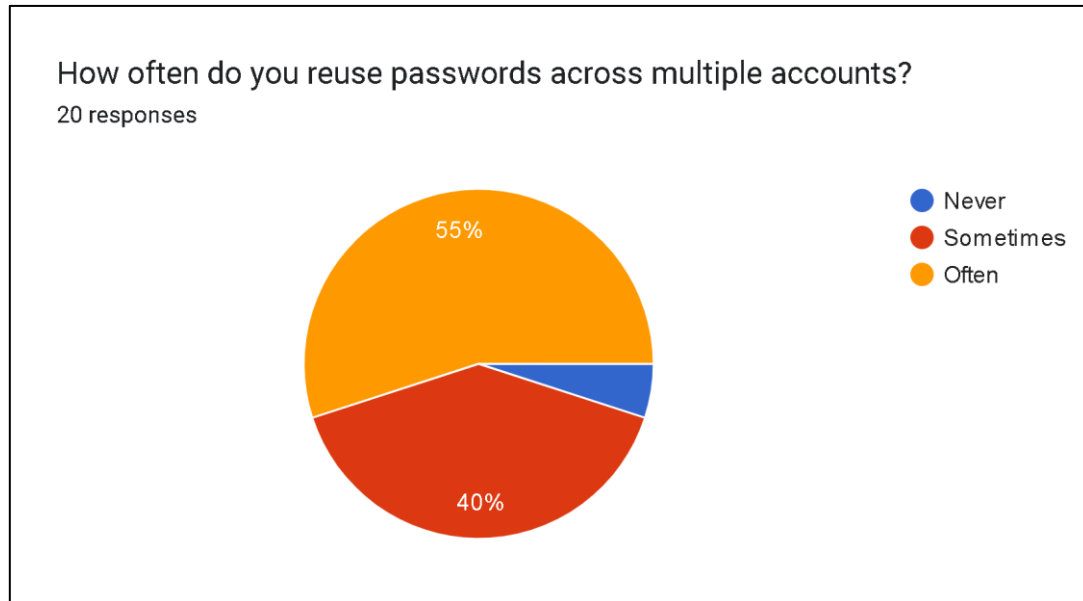


Figure 3.2 Frequency of Password Reuse Across Multiple Accounts

Note. This figure offers a glimpse into the habits surrounding password reuse across multiple accounts. It suggests that for a significant number of users, reusing passwords is a recurring practice, whether frequently or from time to time. This trend points to a potential security vulnerability, emphasizing the importance of adopting stronger password management approaches.

Figure 3.2 illustrates the frequency of password reuse across multiple accounts. The data reveals that 55% of respondents often reuse passwords, while 40% reuse them sometimes, and 5% never reuse passwords. This indicates that while a majority of the respondents are cautious about password reuse, a significant portion still practices it, which may expose them to potential security risks.

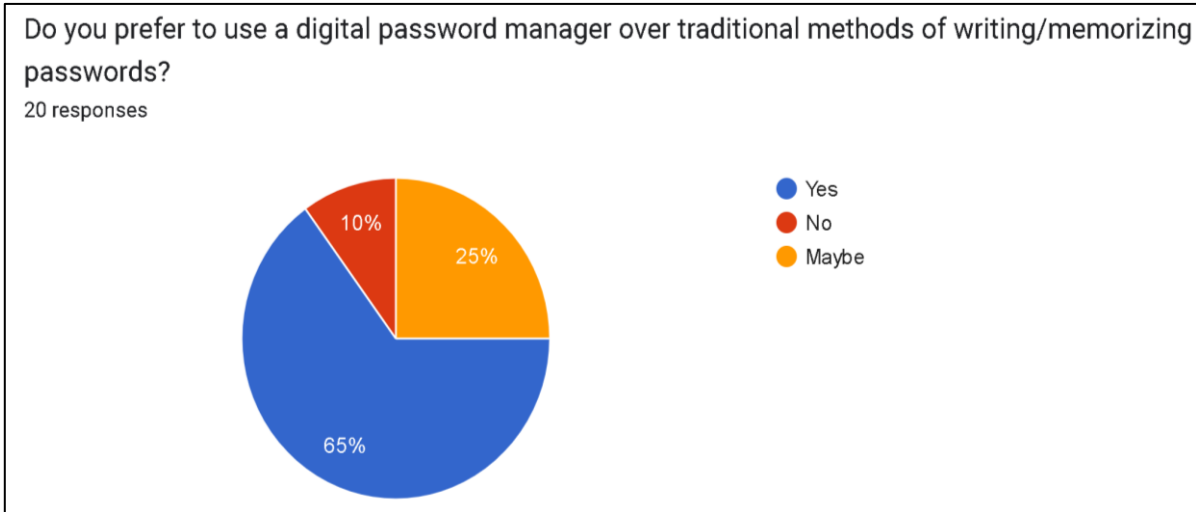


Figure 3.3 Preference to Use a Digital Password Manager

Note. The figure illustrates user preferences regarding the use of digital password managers compared to traditional methods of storing passwords. The responses suggest a tendency toward digital solutions, though some individuals remain uncertain or continue to rely on conventional approaches. This reflects varying levels of adoption and trust in password management tools.

Figure 3.3 shows the preference for using a digital password manager over traditional methods of writing or memorizing passwords. According to the survey, 65% of respondents prefer using a digital password manager, 25% are undecided, and 10% do not prefer using one. This highlights a strong inclination towards digital password managers, suggesting that many respondents see them as a convenient and secure alternative to traditional password management methods.

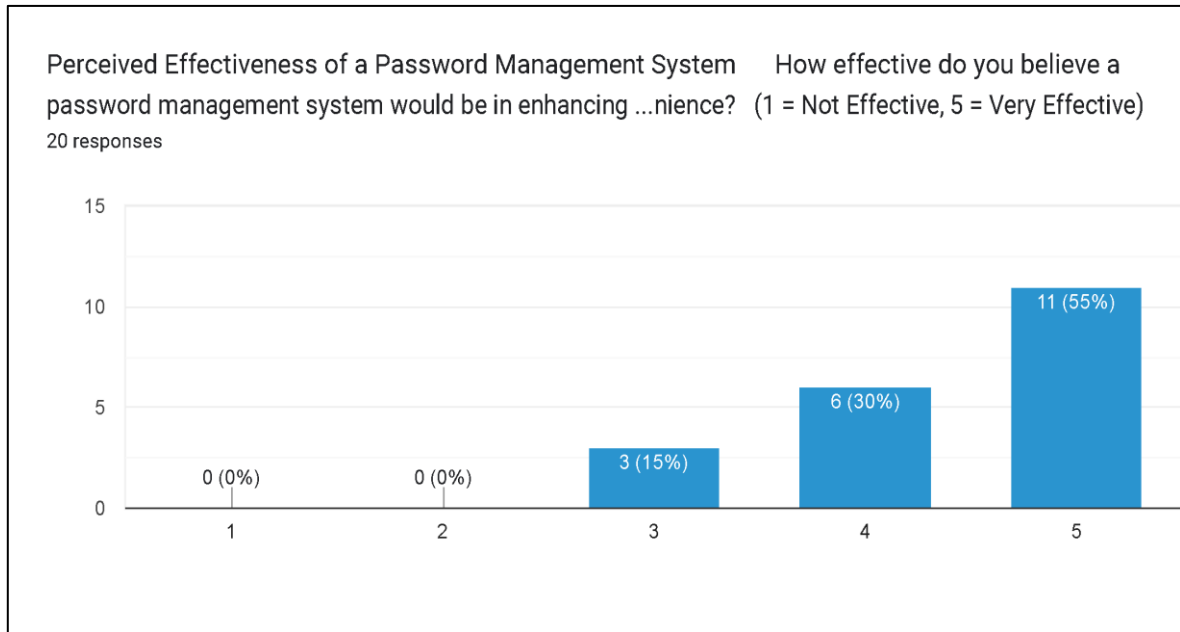


Figure 3.4 Respondent Trust in the Effectiveness of Digital Password Managers

Note. The figure illustrates respondents' views on the effectiveness of digital password managers in enhancing convenience. The distribution of responses indicates a positive inclination toward their usefulness, with higher ratings being more common. This reflects a general perception that such systems can contribute to improved security and accessibility.

Figure 3.4 presents the effectiveness of password management systems in enhancing password security and convenience. The majority of respondents (55%) believe that a password management system would be very effective, while 30% consider it somewhat effective, and 15% are uncertain about its effectiveness. This suggests that most respondents trust password management systems to significantly improve their password security and convenience.

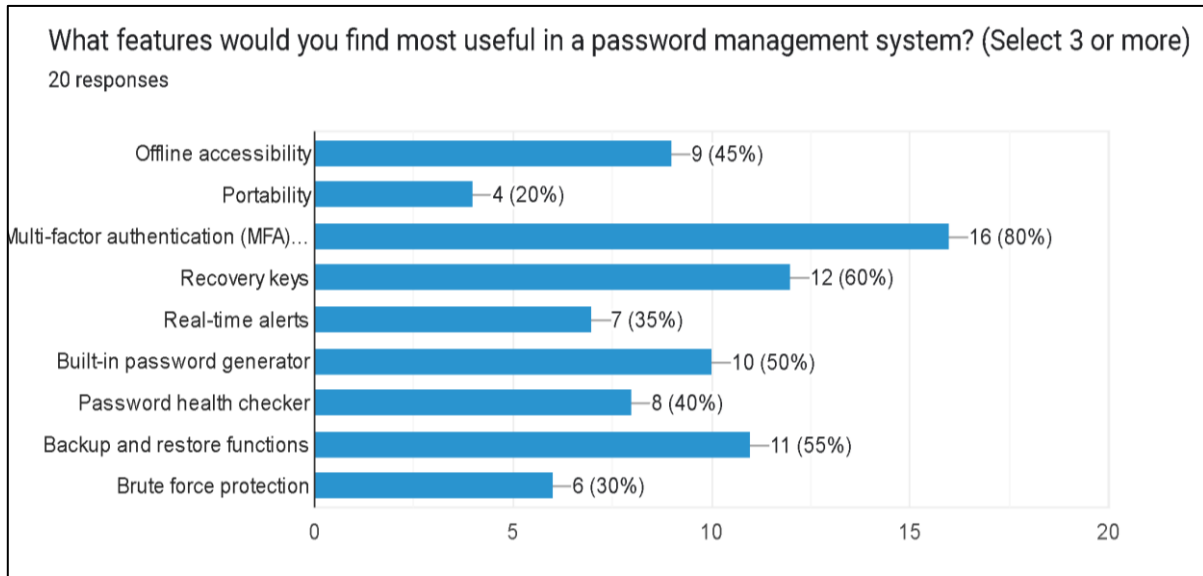


Figure 3.5 Respondent Feature Considerations in Selecting a Password Management System

Note. The figure provides insights into the features respondents find valuable in a password management system. The distribution of responses suggests differing priorities, with some features garnering more interest than others. This reflects varying considerations when selecting a system that balances security and usability.

Figure 3.5 illustrates the features that respondents find most useful in a password management system. 80% of respondents prioritize multi-factor authentication (MFA), while 60% select recovery keys and 55% choose backup and restore functions. This shows that respondents prioritize enhanced security and convenience in a password management system, especially features that provide added layers of protection and flexibility.

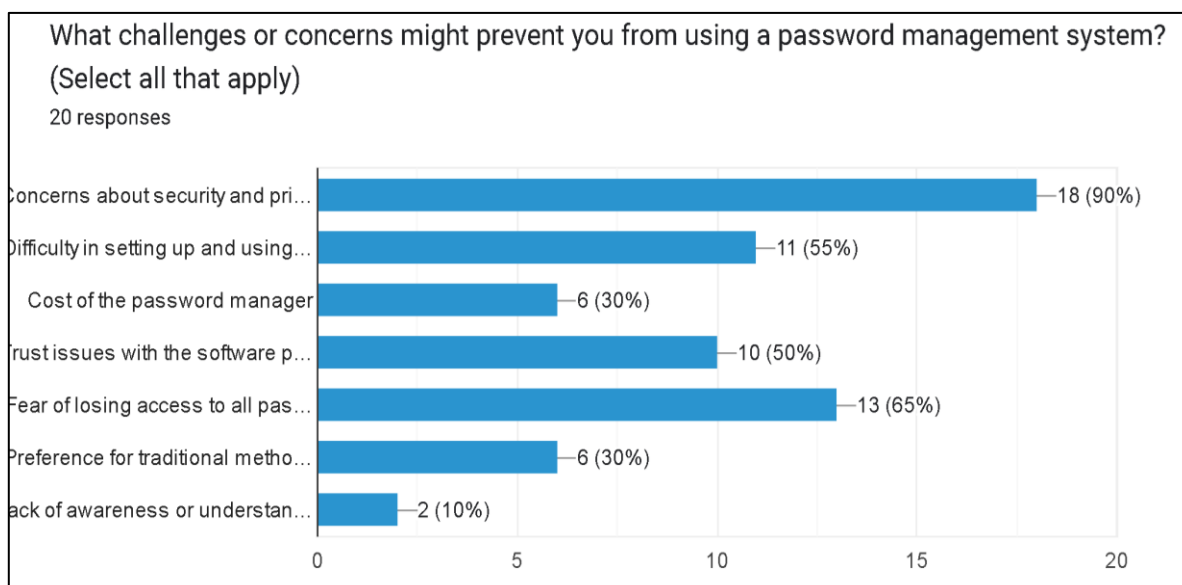


Figure 3.6 Challenges and Concerns Preventing Adoption of Password Management Systems

Note. The figure highlights various factors that may influence the decision to adopt a password management system. These considerations reflect perceptions surrounding usability, reliability, and security, which can impact user confidence in such tools.

Figure 3.6 illustrates the challenges and concerns that might prevent respondents from using a password management system. The most significant barriers include concerns about security and privacy, with 90% of respondents highlighting this issue, followed by the fear of losing access to all passwords if the system fails (65%). Other concerns, such as the difficulty in setting up the system and cost, were also noted by a substantial portion of respondents. This indicates that while respondents recognize the benefits of password management systems, security concerns and system reliability are major deterrents to adoption.

In conclusion, the survey results indicate that respondents are highly concerned about the security and management of their passwords, with a significant number expressing challenges such as remembering passwords and creating strong ones. While a majority show preference for digital password managers, highlighting their convenience and security features, many still face concerns regarding security, privacy, and the potential risks of relying on such

systems. Despite these challenges, the majority of respondents believe that password management systems can enhance both password security and convenience, especially with features like multi-factor authentication, recovery keys, as well as backup and restore functions. The concerns surrounding security, privacy, and system reliability are key factors influencing the willingness to adopt these systems. Overall, the findings suggest that there is a strong demand for password management solutions, but further efforts are needed to address the trust issues and barriers preventing their widespread use.

3.2.2 Software Requirement Specification

This project will be designed and developed using a variety of software tools to ensure the system operates efficiently and responds well. The software tools and technologies used are outlined in Table 3.1.

Table 3.1 List of Software

Software	Description
1) Visual Studio Code	Used for coding and designing the system.
2) Programming Languages - Python, SQL	Used to create the system's interface and manage data.
3) Web Browser	Used to search for online resources and reference materials.
4) Microsoft Office	Used for documentation and report preparation.

Note. This table presents the essential software tools utilized throughout the development process. Each software serves a specific role, ranging from coding and interface design to data management, research, and documentation, contributing to the overall efficiency and organization of the project.

3.2.3 Hardware Requirement Specification

The hardware used to develop the proposed system is outlined in Table 3.2. This setup ensures that the system can run all necessary software and provide the performance required for development.

Table 3.2 List of Hardware

Hardware	Description
• MSI GF63 Thin 9SCXR (Windows 11Laptop) • Specification: ✓ Intel Core i7 9750H 2.60GHz ✓ 16GB RAM ✓ 1TB SSD	Used for developing the system.

Note. This table outlines the hardware utilized for system development. The listed laptop, equipped with a high-performance processor, ample memory, and large storage, provided the necessary computing power to support coding, testing, and overall project execution efficiently.

3.3 System Design

In this phase, the gathered requirements are analyzed, and the system design is created to ensure secure password storage and retrieval. The design follows a systematic approach that includes the conceptual design, which covers key diagrams such as the use case diagram, sequence diagram, class diagram, and flowchart, and the actual design, which describes the user interface design for ease of use. The design prioritizes security, efficiency, and usability to meet the system's objectives.

3.3.1 Use Case Diagram

The use case diagram is used to clearly define and understand how users will interact with the system. It helps outline the key functions of the proposed password management system prototype, ensuring that the design meets user needs and expectations. By mapping out the different tasks a user can perform, the diagram offers a clear overview of the system's capabilities and user interactions.

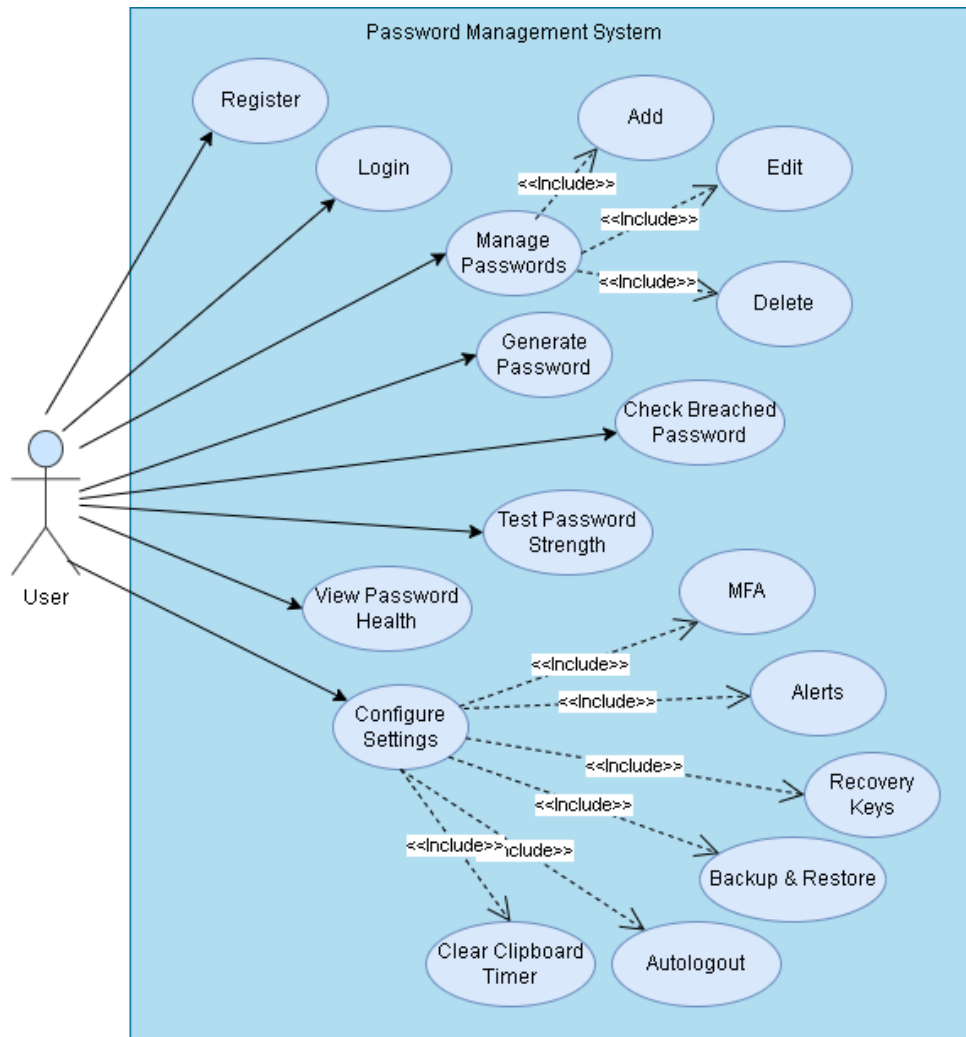


Figure 3.7 Use Case Diagram of Password Management System

Note. This figure outlines the key interactions within the Password Management System, depicting user actions and system functionalities.

Figure 3.7 presents the Use Case Diagram for the prototype, illustrating the interactions between the "User" and the "Password Management System." The user can perform key functions such as registering, logging in, and managing passwords (adding, editing, or deleting). Additional actions include generating new passwords, testing password strength, and checking if a password has been breached. Users can also view password health status and configure system settings such as multi-factor authentication (MFA), alerts, and backup and restore

options. The system, in turn, is responsible for handling critical backend processes including encrypting, decrypting, securely storing passwords, and managing user configurations.

3.3.2 Sequence Diagram

The sequence diagram is an essential tool for conceptual design as it visualizes the step-by-step interactions between the user and the system, ensuring the process flow is logical and meets the intended requirements. By outlining the sequential order of operations for various tasks, it helps identify potential issues, clarifies the system's behavior, and ensures all functionalities are implemented correctly. For the password management prototype, the sequence diagrams provide a detailed view of core processes, including registration, login, password management, and other key features. Each diagram, illustrated in Figures 3.8 to 3.14, focuses on a specific function, ensuring clarity and alignment with user expectations.

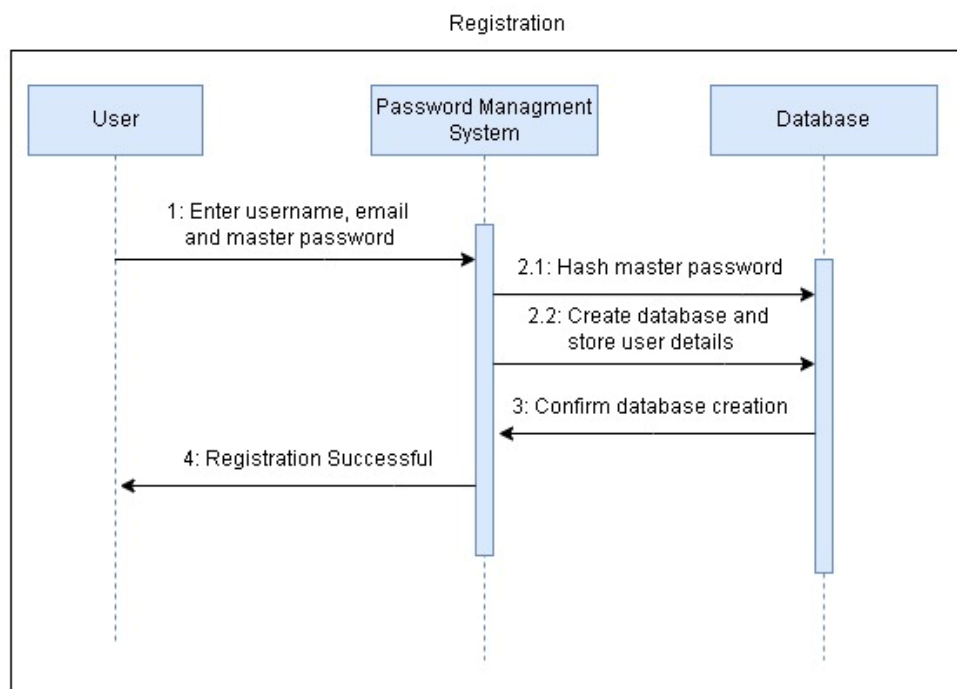


Figure 3.8 Sequence Diagram for Registration Process

Note. This sequence diagram illustrates the registration flow, highlighting interactions between the user, system, and database. It outlines key steps, including password hashing and data storage, ensuring secure account creation.

Figure 3.8 shows the sequence diagram for the registration process, where the user initializes the system by creating a master password. The user enters the master password, which is hashed by the system using PBKDF2. The hashed password is used to securely create and encrypt an SQLite database, ensuring safe storage for future entries. Once the database is created, the system confirms the setup by displaying "Registration Successful." This sequence ensures secure initialization of the password management system.

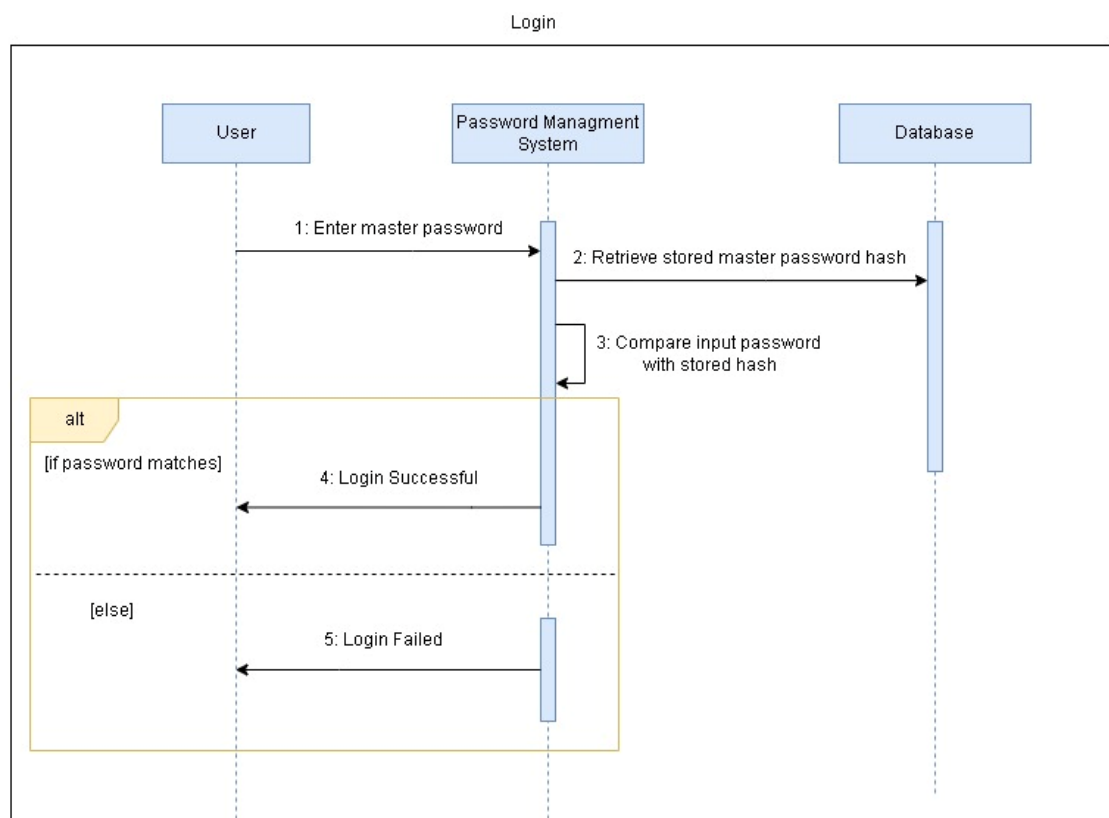


Figure 3.9 Sequence Diagram for Login Process

Note. This sequence diagram illustrates the login process, showing interactions between the user, system, and database. It details how the system retrieves and verifies the stored password hash before granting or denying access based on the input match.

Figure 3.9 illustrates the sequence diagram for the login process, detailing how the system verifies the user's credentials. The user submits their master password, and the system

retrieves the hashed password from the SQLite database. The system compares the submitted password with the stored hash. If they match, the user is granted access with a "Login Successful" message. If not, a "Login Failed" message is displayed. This sequence ensures secure and reliable authentication for accessing the system.

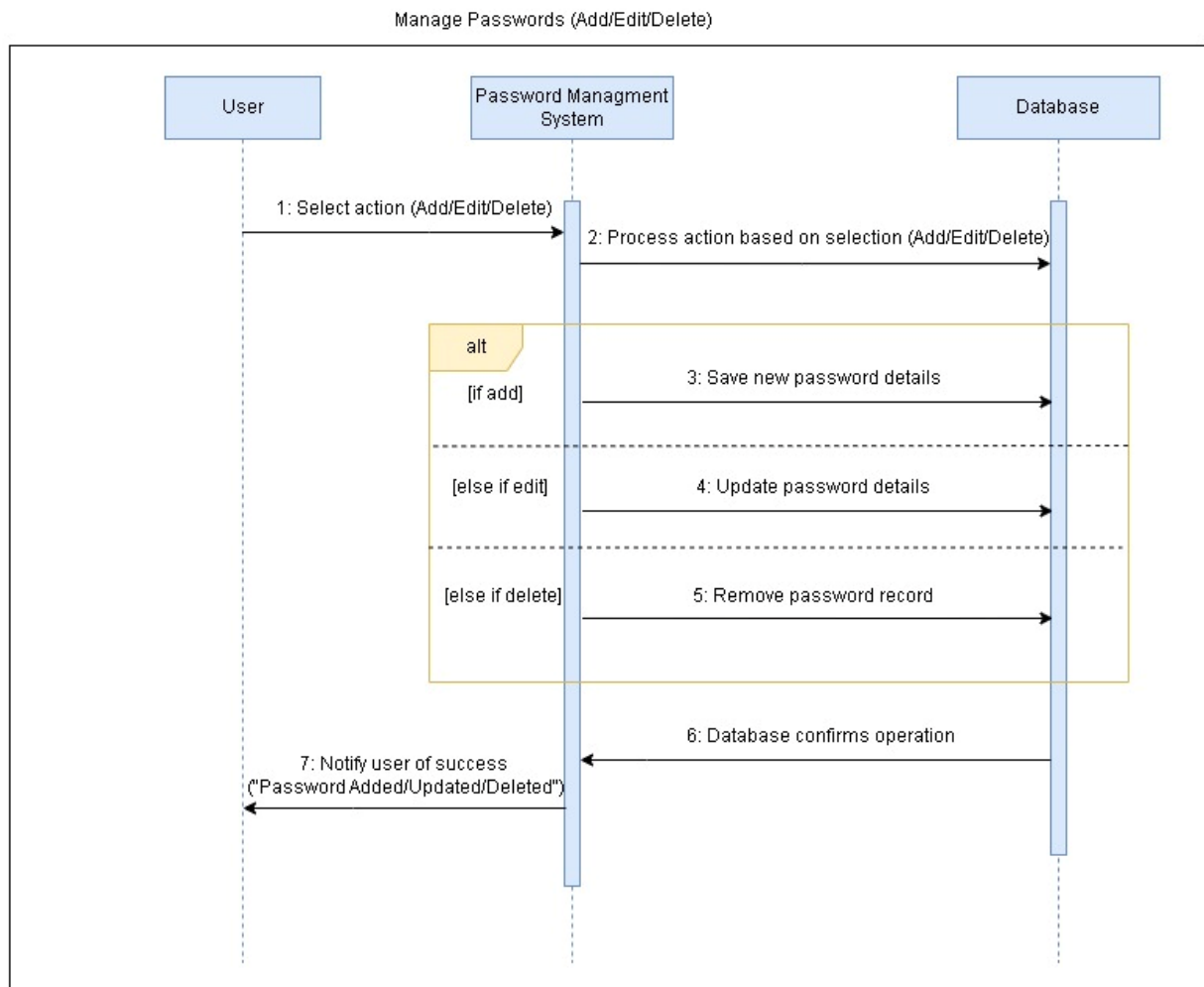


Figure 3.10 Sequence Diagram for Manage Passwords (Add/Edit/Delete) Process

Note. This sequence diagram illustrates the process of managing passwords (Add/Edit/Delete) in the password management system. It depicts how the user selects an action, the system processes the request, and the database performs the corresponding operation. The system then notifies the user of the successful completion of the action.

Figure 3.10 depicts the sequence diagram for managing passwords, which includes adding, editing, and deleting password to enable efficient and secure management of stored credentials:

- **Add a Password:** The user inputs details such as username, password, and platform. The system stores these details in the SQLite database and confirms the addition with a "Password Added" message.
- **Edit a Password:** The user selects an existing entry, and the system retrieves the current details. After the user provides updates, the system saves the changes and confirms them with a "Password Updated" message.
- **Delete a Password:** The user selects an entry to delete, and the system removes the record from the database, confirming the deletion with a "Password Deleted" message.

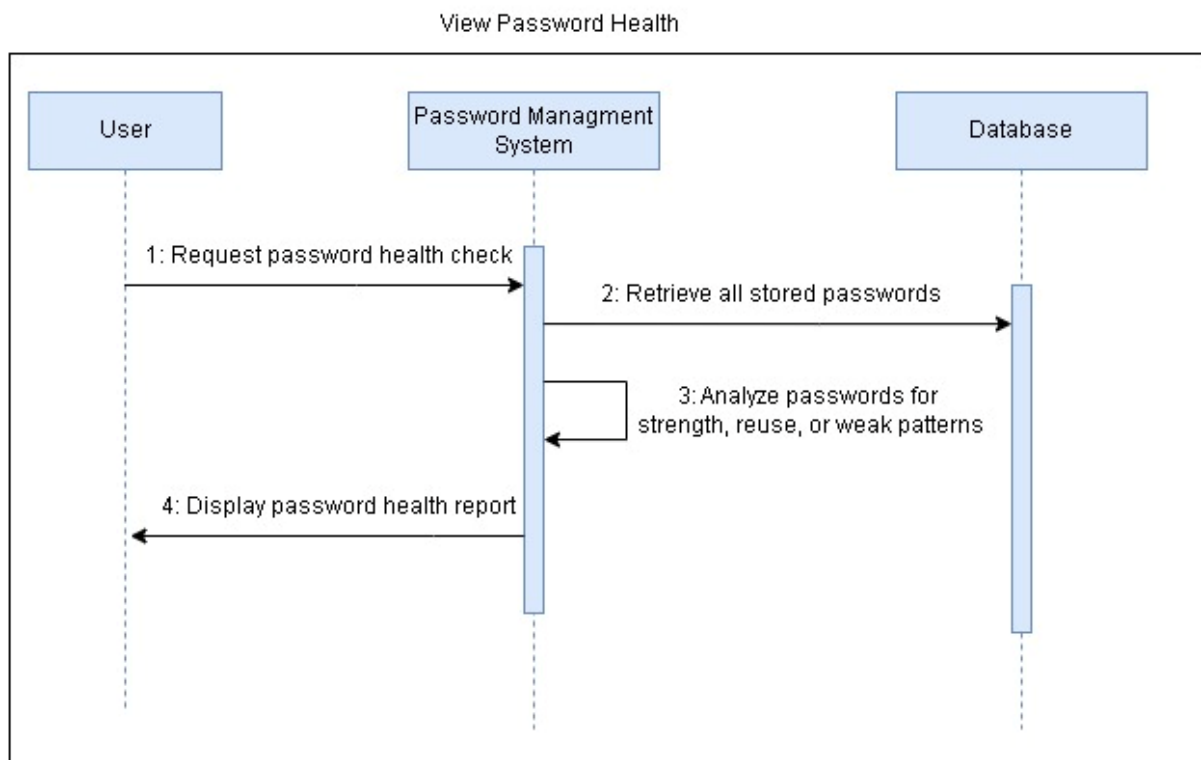


Figure 3.11 Sequence Diagram for View Password Health Process

Note. This sequence diagram illustrates the process of viewing password health in the password management system. It details how the user requests a password health check, the system retrieves stored

passwords from the database, and then analyzes them for strength, reuse, or weak patterns. Finally, the system generates and displays a password health report to the user.

Figure 3.11 shows the sequence diagram for viewing password health, allowing the user to assess the strength of their stored passwords. The user initiates a password health check, prompting the system to retrieve all stored passwords from the SQLite database. The system analyzes the passwords for strength, reuse, or weak patterns and displays a detailed password health report. This sequence helps users maintain strong and secure passwords.

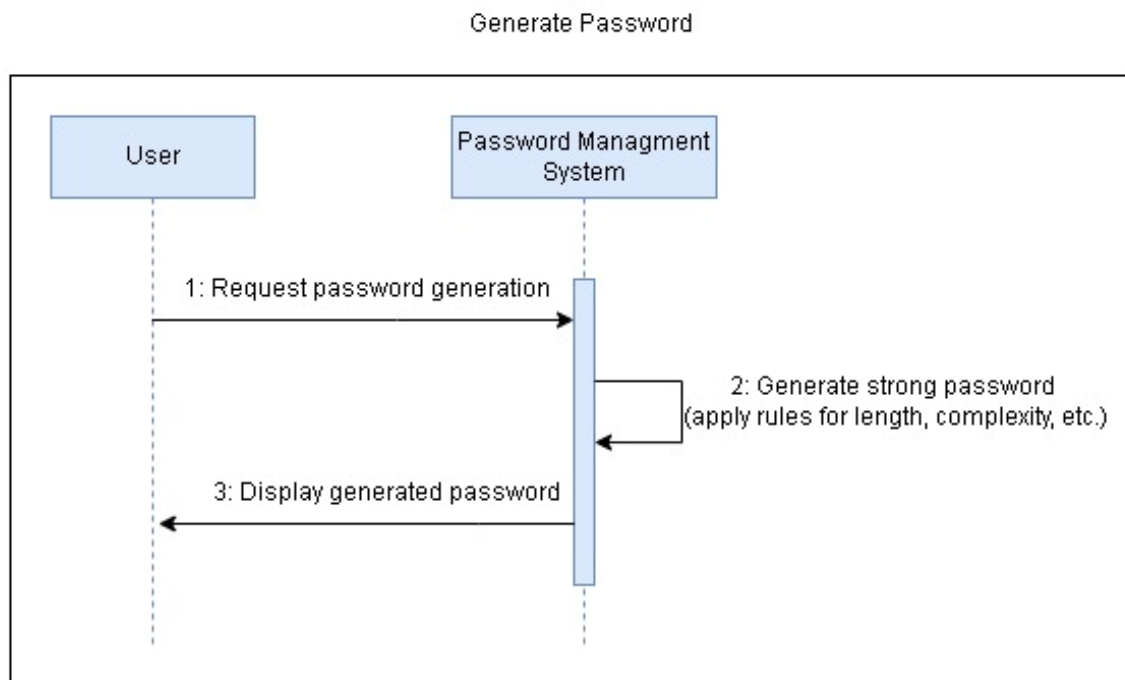


Figure 3.12 Sequence Diagram for Generate Password Process

Note. This sequence diagram illustrates the process of generating a password in the password management system. The user initiates the process by requesting password generation. The system then applies predefined rules for length, complexity, and randomness to generate a strong password. Finally, the generated password is displayed to the user.

Figure 3.12 illustrates the sequence diagram for the password generation process. The user requests a new password, and the system generates a strong password based on predefined

rules such as length and complexity. The generated password is displayed to the user. This feature helps users adopt better password practices by generating secure passwords.

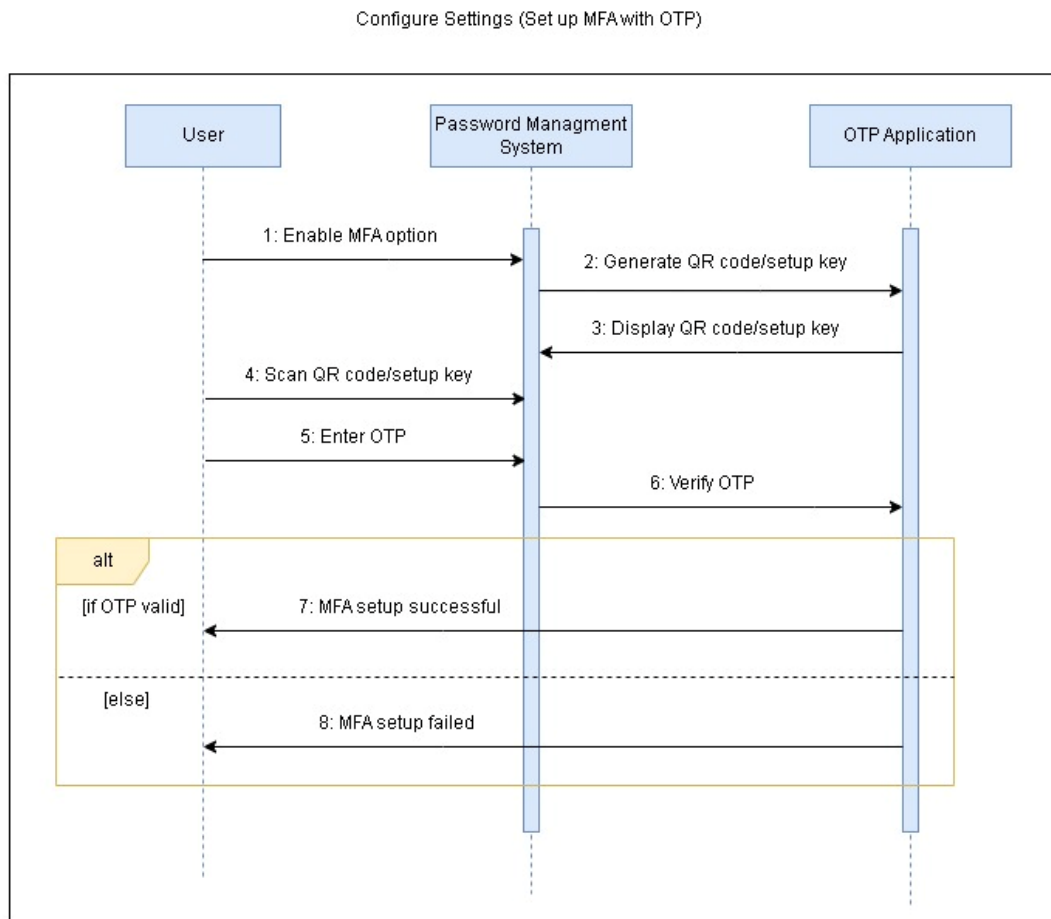


Figure 3.13 Sequence Diagram for Configure Setting (MFA with OTP) Process

Note. This sequence diagram represents the interactions between the user, the password management system, and an OTP application. It highlights the communication flow required to establish an additional security layer, ensuring enhanced authentication measures. The diagram also includes conditional outcomes based on verification results.

Figure 3.13 shows the sequence diagram for setting up Multi-Factor Authentication (MFA) using a One-Time Password (OTP) which provides an additional layer of security. This process involves interactions between the user, the password management system, and the OTP application:

- The user enables the MFA option in the system settings.

- The system generates an OTP request and prompts the user to retrieve the OTP from their designated OTP application.
- The user enters the OTP into the system.
- The password management system verifies the OTP:
 - If the OTP is valid, the system confirms the setup of MFA with a success message.

If the OTP is invalid, the system notifies the user that the setup has failed.

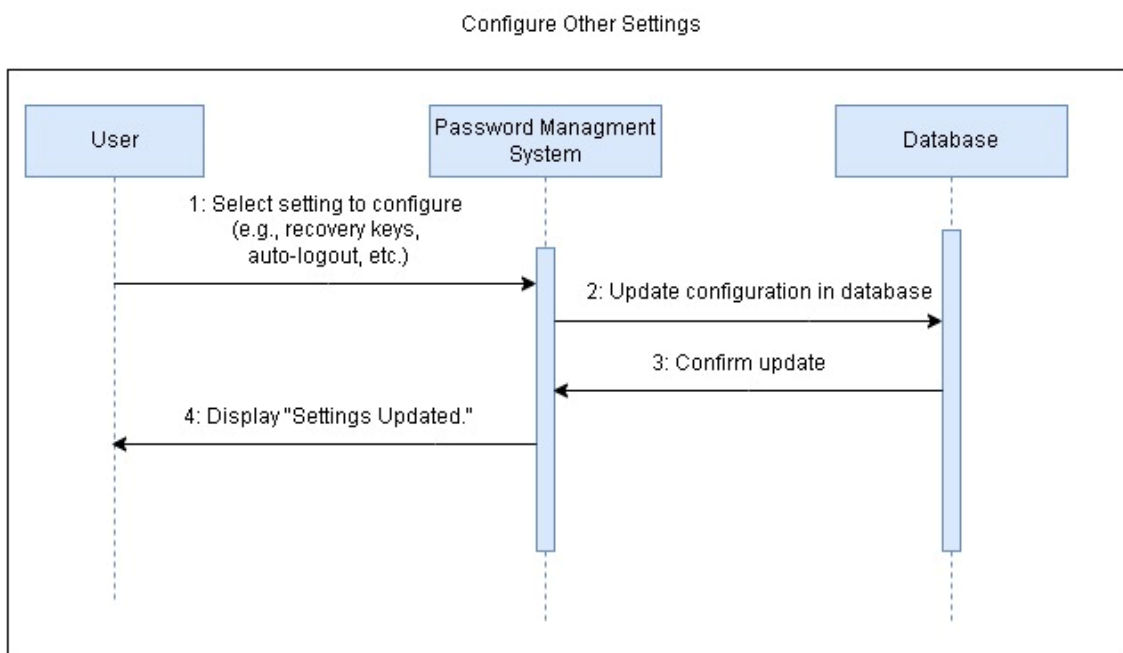


Figure 3.14 Sequence Diagram for Configure Other Settings Process

Note. This sequence diagram illustrates the interaction between the user, the password management system, and the database. It represents the steps involved in modifying system preferences, ensuring that user-defined configurations are updated and stored successfully. The diagram highlights the communication flow and confirmation mechanisms for maintaining system settings.

Figure 3.14 illustrates the sequence diagram for configuring other system settings, such as recovery keys, auto-logout, and clipboard timers which ensures that the system can be tailored to the user's preferences, enhancing usability and convenience while maintaining security:

- The user selects the setting they want to configure from the options available in the password management system.
- The system updates the chosen configuration in the SQLite database.
- The database confirms the update to the password management system.
- The password management system notifies the user that the settings have been successfully updated with a "Settings Updated" message.

3.3.3 Class Diagram

A class diagram is used for the conceptual design of the password management prototype to define the structure of the system and establish relationships between its components. It provides a clear, visual representation of the system's core classes—such as User, Password, Settings, and OTP—along with their attributes, methods, and associations. By organizing the system into logical components with well-defined relationships, the class diagram helps ensure consistency in design, supports modular development, and facilitates the understanding of system functionality. The diagram highlights key functionalities like password management, settings configuration, and multi-factor authentication, aiding in planning, implementation, and communication of the system's conceptual design. Figure 3.15 shows the class diagram for the prototype.

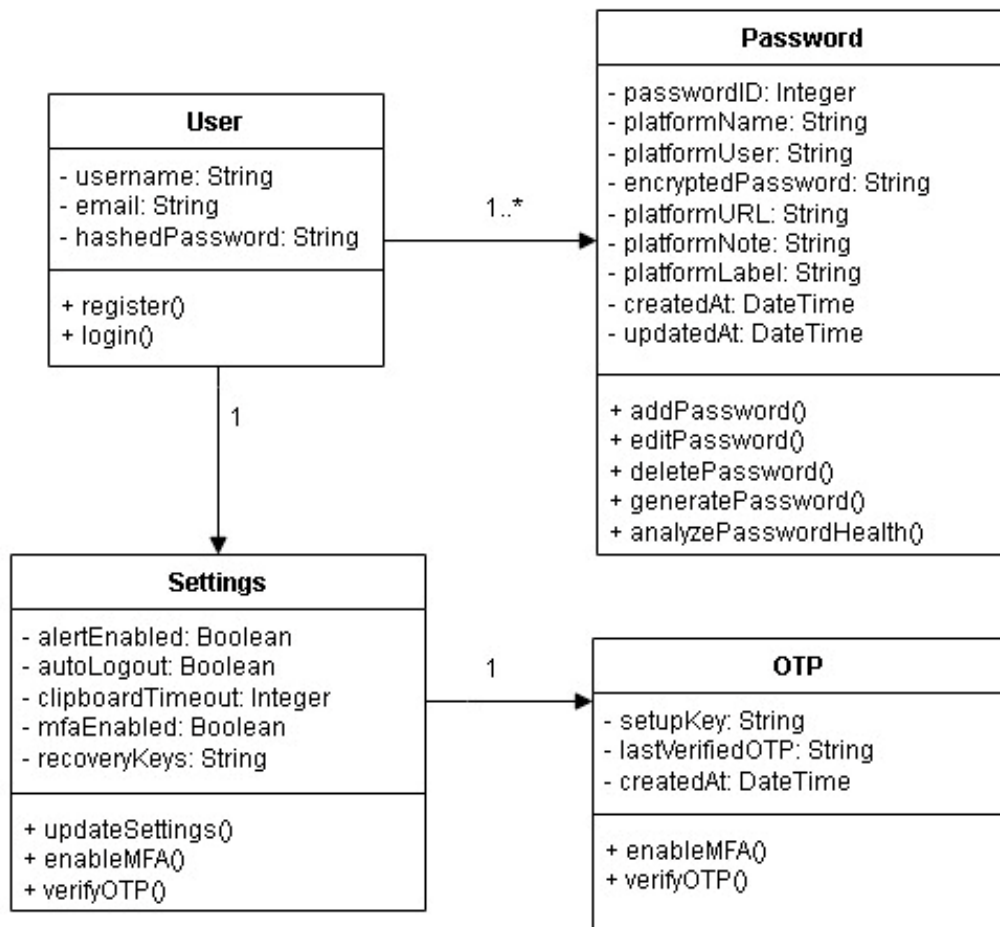


Figure 3.15 Class Diagram

Note. This class diagram illustrates the structure and relationships between key components in the password management system. It defines how users interact with stored passwords, system settings, and MFA. The diagram highlights entity attributes and functions, emphasizing secure password storage, user-defined configurations, and verification mechanisms to enhance security and usability.

From Figure 3.15, the relationships between the classes are as follows:

- **User and Password:** A **1-to-many** relationship, where one user can have multiple stored passwords. This indicates that a user can manage several passwords securely within the system.
- **User and Settings:** A **1-to-1** relationship, where each user has a unique settings profile. This profile includes configurations like auto-logout, clipboard timeout, and MFA status.

- **Settings and OTP:** A **1-to-1** relationship, where the settings for each user are linked to an OTP setup. This supports MFA functionality by associating the settings directly with OTP operations such as enabling MFA and verifying OTP codes.

These relationships help define the interactions between the system's components and contribute to a well-organized design.

3.3.4 Flowchart

A flowchart is used for the conceptual design of this password management prototype to visually represent the sequence of steps involved in securely storing and retrieving passwords. It provides a clear and structured outline of how the system works, making it easier to understand the flow of data and the operations involved in encryption and decryption. The flowchart ensures that all security measures, such as AES-GCM encryption and PBKDF2 key derivation, are applied correctly and sequentially, minimizing the risk of exposing sensitive data.

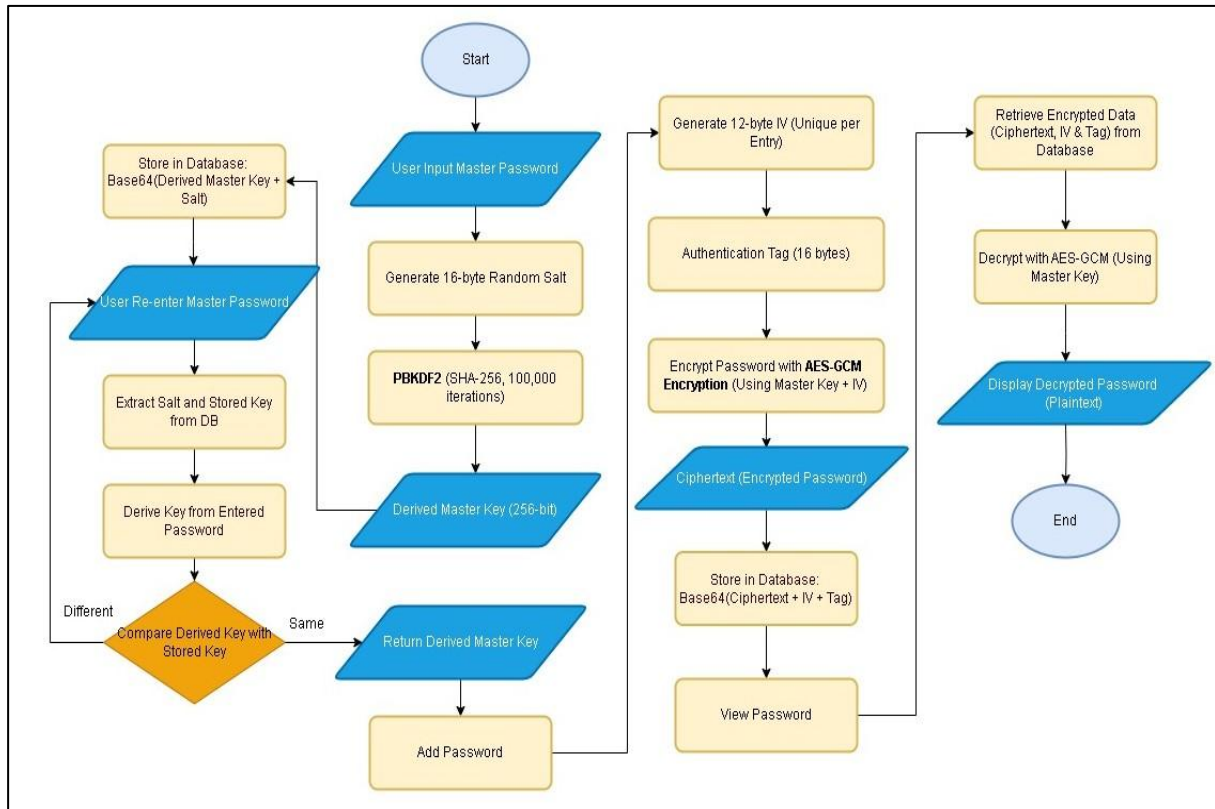


Figure 3.16 Flowchart of How Passwords are Securely Stored and Retrieved

Note. This flowchart illustrates the process of securely storing and retrieving passwords in the password management system. It outlines the encryption and decryption mechanisms using AES-GCM, ensuring both data confidentiality and integrity.

Figure 3.16 shows the flowchart of how passwords are securely stored and retrieved using AES-GCM and PBKDF2. When a user sets up or logs in with their master password, the system first generates or retrieves a 16-byte random salt stored in the database. This salt, combined with the master password, is processed through PBKDF2 using SHA-256 and 100,000 iterations to derive a 256-bit master key, which strengthens the password against brute-force and rainbow-table attacks. The derived key and salt are then Base64-encoded and stored. During authentication, the system re-derives the key from the entered password and compares it with the stored key; a match grants access to the master key.

When adding a new password, the system uses AES-GCM encryption: the password is encrypted using the derived master key and a unique 12-byte initialization vector (IV), producing a ciphertext and a 16-byte authentication tag to ensure data integrity. These components—ciphertext, IV, and tag—are then Base64-encoded and stored in the database.

When the user views a stored password, the system performs decryption using AES-GCM: it retrieves the encrypted data, verifies the tag's authenticity, and decrypts the ciphertext using the master key and the original IV.

By combining PBKDF2's key-strengthening capabilities with AES-GCM's confidentiality and integrity guarantees—alongside unique salts and IVs to prevent ciphertext reuse—the system ensures that even if the database is compromised, attackers cannot feasibly decrypt the data without the correct master password.

3.3.5 User Interface Design

The user interface design of the Password Management System prototype, shown in Figures 3.17 to 3.21, includes various menus and forms to ensure ease of use while providing a secure and comprehensive experience for managing passwords.

 Home Password Health Settings 								
	Platform	Label	Username	Password	URL	Notes	Date Added	Date Modified
	Google	Website	raphael	*****	https://www.google.co.uk/	-	8/1/2025	9/1/2025
								
								

Figure 3.17 Main Menu for Password Management System Prototype

Note. This figure shows the main menu of the Password Management System prototype, featuring a structured table for managing stored credentials with options to edit or add entries.

Figure 3.17 illustrates the main menu of the Password Management System, acting as the primary interface for users. This menu includes essential navigation options such as the Home button, links to Password Health and Settings, and an Add button for creating new password entries. Below these elements, a table presents existing password entries, showcasing details like the platform, label, username, password, URL, notes, date added, and date modified, with an Edit button available for each entry.

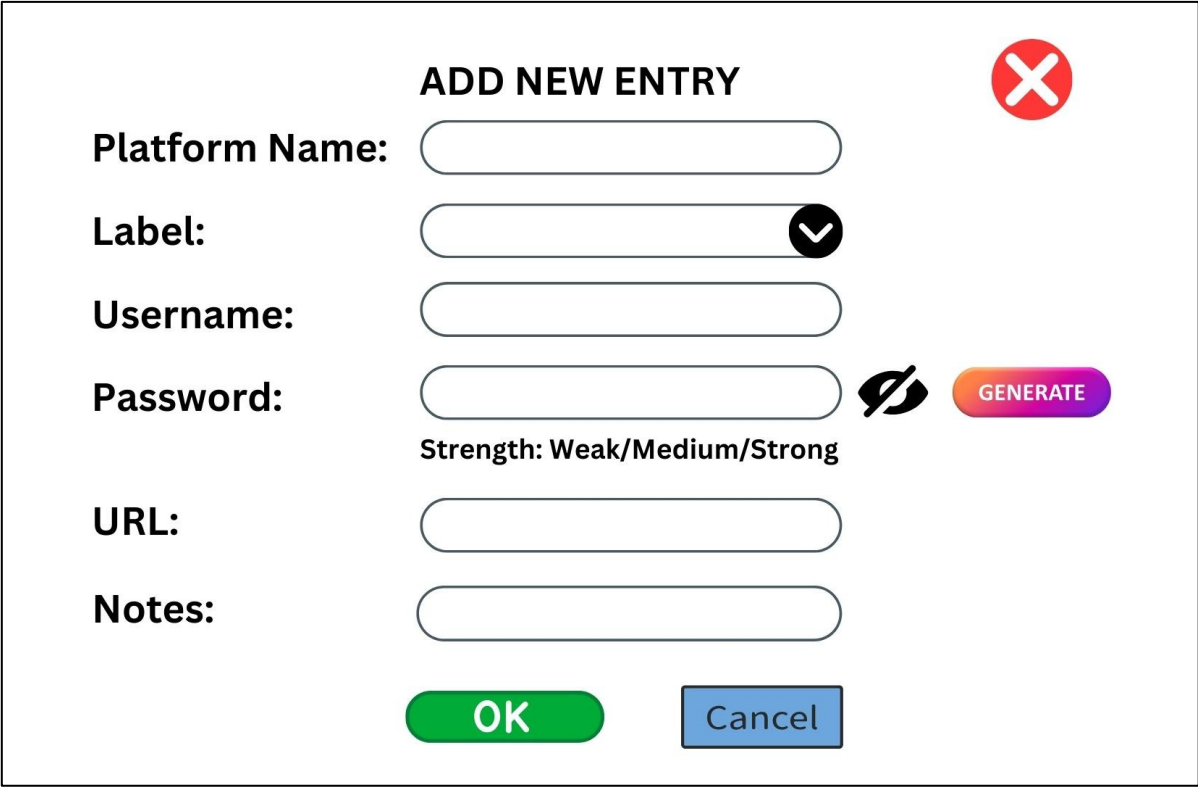
The image shows a 'Add New Entry' form with a title bar at the top. The title bar contains the text 'ADD NEW ENTRY' on the left and a red circular close button with a white 'X' on the right. Below the title bar, the form has several input fields: 'Platform Name:' followed by a text input field; 'Label:' followed by a dropdown menu with a downward arrow; 'Username:' followed by a text input field; 'Password:' followed by a text input field, a visibility toggle icon (an eye with a slash), and a 'GENERATE' button with a gradient background; 'URL:' followed by a text input field; and 'Notes:' followed by a text input field. Below these fields are two buttons: a green 'OK' button and a blue 'Cancel' button. The text 'Strength: Weak/Medium/Strong' is positioned below the password input field.

Figure 3.18 Add New Entry Form

Note. This figure displays the "Add New Entry" form in the Password Management System prototype, enabling users to securely store credentials with features like password strength indication, generation, and visibility toggle.

As shown in Figure 3.18, the Add New Entry Form provides users with the interface to input new passwords. The form contains fields for the platform, a dropdown to select the label, username, password (with a show/hide option and password strength indicator), a generate password button, as well as URL and notes fields. The user can finalize the entry by selecting OK, cancel the operation with the Cancel button, or exit the form using the Close button.

EDIT ENTRY

Platform Name:

Label:

Username:

Password: **GENERATE**

Strength: Weak/Medium/Strong

URL:

Notes:

OK **Cancel**

Figure 3.19 Edit Entry Form

Note. This figure illustrates the edit entry form in the Password Management System, allowing users to modify stored credentials while maintaining security and accessibility.

Figure 3.19 depicts the Edit Password Form, which is used to modify existing passwords. Similar to the Add Password Form, it comes pre-filled with the current password information, allowing the user to update the platform, label, username, password, and other details. The user can either save the changes by clicking OK or cancel them using the Cancel button.

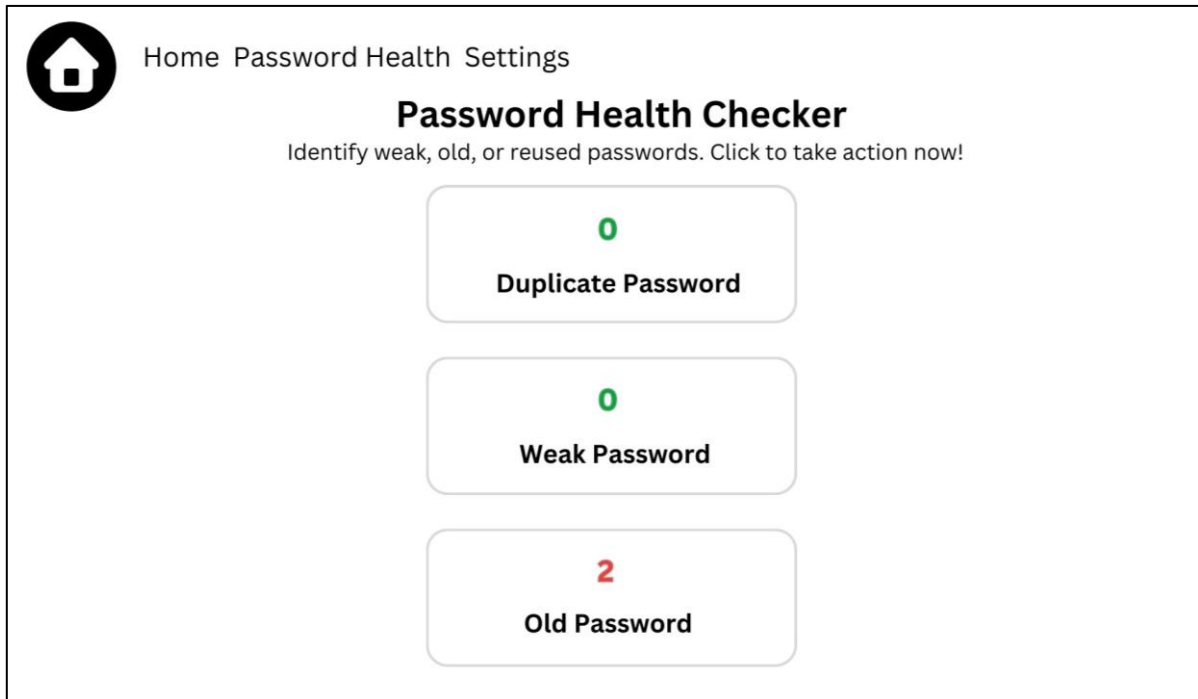


Figure 3.20 Password Health Checker Menu

Note. This figure illustrates the Password Health Checker menu, which helps users identify and manage duplicate, weak, or outdated passwords to enhance account security.

As represented in Figure 3.20, the Password Health Checker Menu allows users to review the status of their passwords. This menu highlights any duplicated, weak, or old passwords, presenting a visual summary of potential issues. Users can then select any problem password to take corrective action, such as updating or replacing outdated or vulnerable passwords.

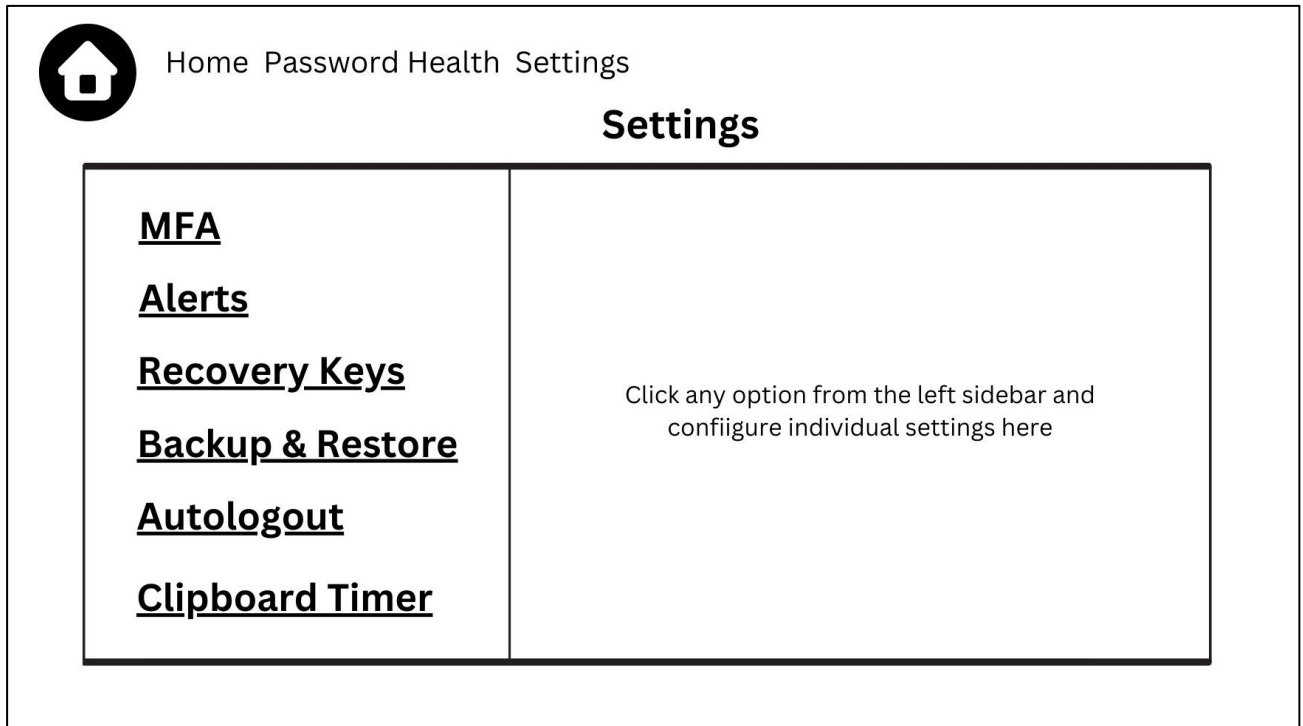


Figure 3.21 Settings Menu

Note. This figure displays the Settings Menu, where users can configure various security features such as MFA, alerts, recovery keys, backup and restore, autologout, and clipboard timer to enhance account protection and usability.

Figure 3.21 displays the Settings Menu, where users can adjust various system configurations. The options include enabling Multi-Factor Authentication (MFA), adjusting auto-logout preferences, and managing backup and other settings. This menu provides users with the flexibility to tailor their password management experience, enhancing both security and usability.

3.4 Summary

In summary, this chapter has provided a detailed explanation of the methodology used, which serves as the foundation for developing the proposed Password Management System Prototype. It also highlighted the process of gathering requirements, ensuring a comprehensive

understanding of what is needed to build the prototype effectively. The purpose of collecting and analyzing this information is to ensure that all necessary requirements for developing the system are identified and addressed. The output of this chapter serves as a reference for the next phase of the Waterfall Model, which is the implementation phase. By identifying the target users and understanding their needs, the development of the prototype can be guided in the right direction. The system requirements and system design phases within the Waterfall methodology form the structural backbone for analyzing and developing the system. Key diagrams, including the use case diagram, sequence diagram, and class diagram, were created to provide a clear and detailed representation of the system's functionality, workflow, and structure. These diagrams collectively help visualize the interactions between users and the system, the flow of processes, and the relationships between different components. Additionally, the proposed software and hardware requirements outlined in this chapter will serve as valuable references during the development phase.

CHAPTER 4: IMPLEMENTATION

4.1 Overview

This chapter presents **Phase 3 of the Waterfall methodology: Implementation** of the Offline Password Management System developed using Python. It highlights the system's architecture, tools used, encryption mechanisms, password security estimations, real-world attack simulations, and user interface. The focus is to demonstrate how theoretical designs from Chapter 3 are translated into a secure, functioning desktop application capable of encrypting, storing, and analyzing passwords offline.

4.2 Implementation of the Password Management System

This section outlines the implementation process of the Password Management System, covering hardware specifications, Integrated Development Environment (IDE), database setup, library installation, and the key coding involved in the development.

4.2.1 Hardware Specification

The system was developed using a laptop with the following specifications:

- **Processor:** Intel Core i7 9750H 2.60GHz (6 cores, 12 threads)
- **RAM:** 16GB DDR4
- **Storage:** 1TB SSD
- **Operating System:** Windows 11
- **Graphics:** NVIDIA GeForce GTX 1650 With Max-Q Design
- **Display:** 15.6-inch Full HD (1920 x 1080)

These hardware specifications ensured smooth execution of the system during the development phase. The laptop's robust performance allowed for efficient development and testing, with sufficient memory and storage for handling various tasks, including database management, GUI development, and cryptographic operations.

4.2.2 Integrated Development Environment (IDE)

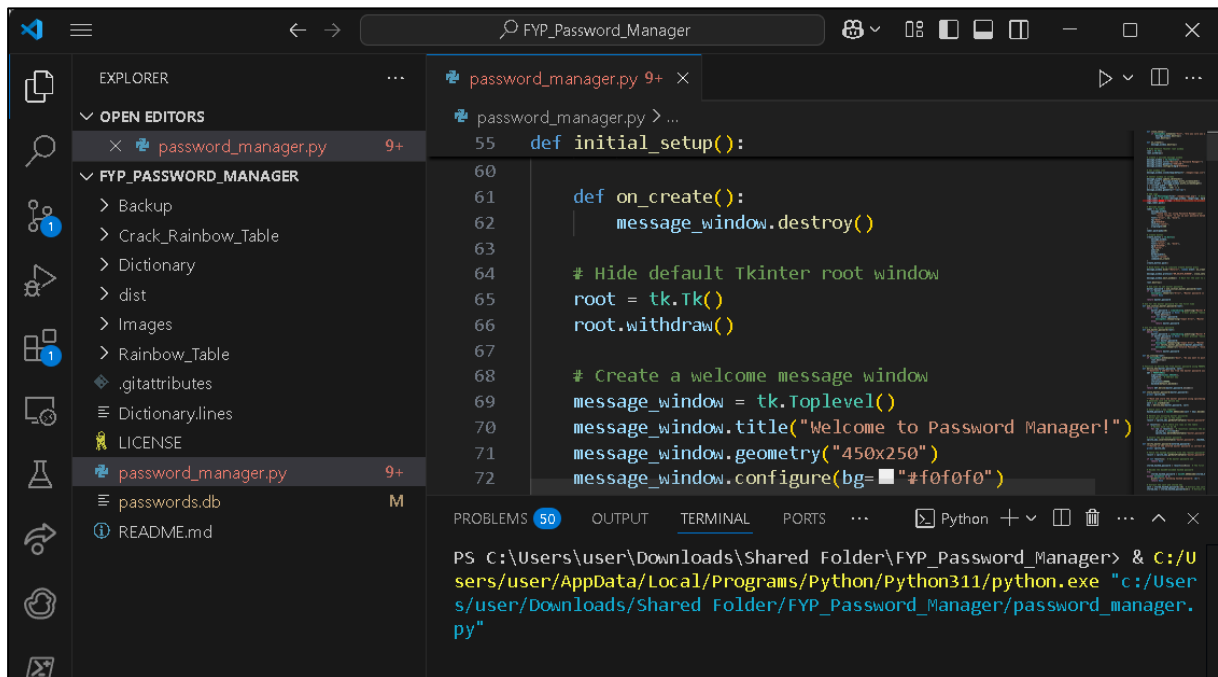


Figure 4.1 Visual Studio Code Interface for System Development

Note. This figure displays the Visual Studio Code (VS Code) interface used for developing the Password Management System, showcasing the code editor, integrated terminal, and Python extension for enhanced coding, linting, and auto-completion.

The system was implemented using Visual Studio Code, a popular, lightweight code editor by Microsoft. VS Code offers various features like integrated debugging, syntax highlighting and an extensive extension marketplace, which greatly facilitated the development process. Figure 4.1 shows the interface of VS Code, where the source code for the password management system was written and executed. The Python extension was installed to enhance the IDE with

features like code linting, auto-completion, and environment management. Here are the steps required to set up and use VS Code for developing the Password Management System:

1. Download and install VS Code:

- Visit the official website: <https://code.visualstudio.com/Download>
- Download the installer for your operating system and follow the installation instructions.

2. Install the Python extension:

- Open VS Code and go to the **Extensions** view by clicking on the Extensions icon in the Activity Bar on the side of the window.
- Search for the **Python** extension in the marketplace and click **Install**.
- This extension provides features like syntax highlighting, IntelliSense (auto-completion), and Python-specific debugging tools.
- **Figure 4.2** below shows how the Python extension appears in the VS Code marketplace.

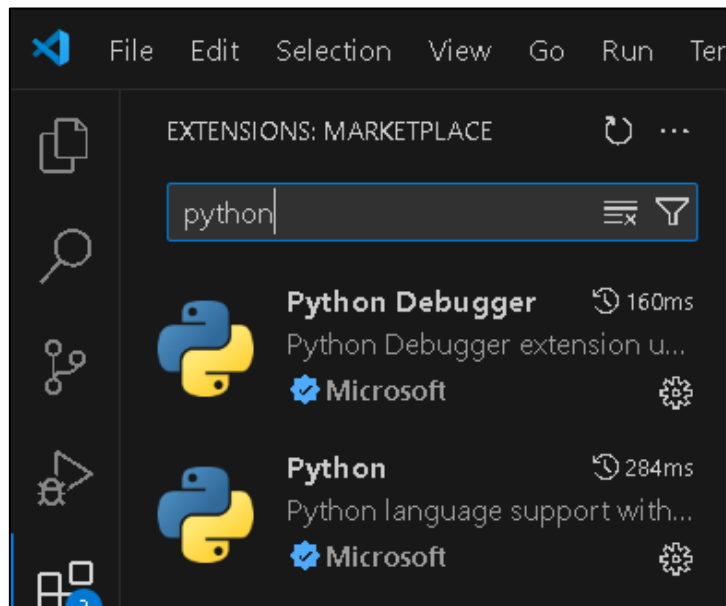


Figure 4.2 Python Extension in the VS Code Marketplace

Note. This figure shows how the Python extension appears in the VS Code marketplace, where users can install it to enable features such as code linting, auto-completion, and environment management for Python development.

3. Install required libraries:

- Install the necessary Python libraries for the project, such as **cryptography**, **pyotp**, **pyperclip**, etc., by running the following command in the VS Code terminal:

```
pip install library_name
```

Once these steps are completed, your VS Code environment will be fully set up for developing and testing the Password Management System. The combination of Python and VS Code ensures a productive and efficient development process.

4.2.3 Database Setup

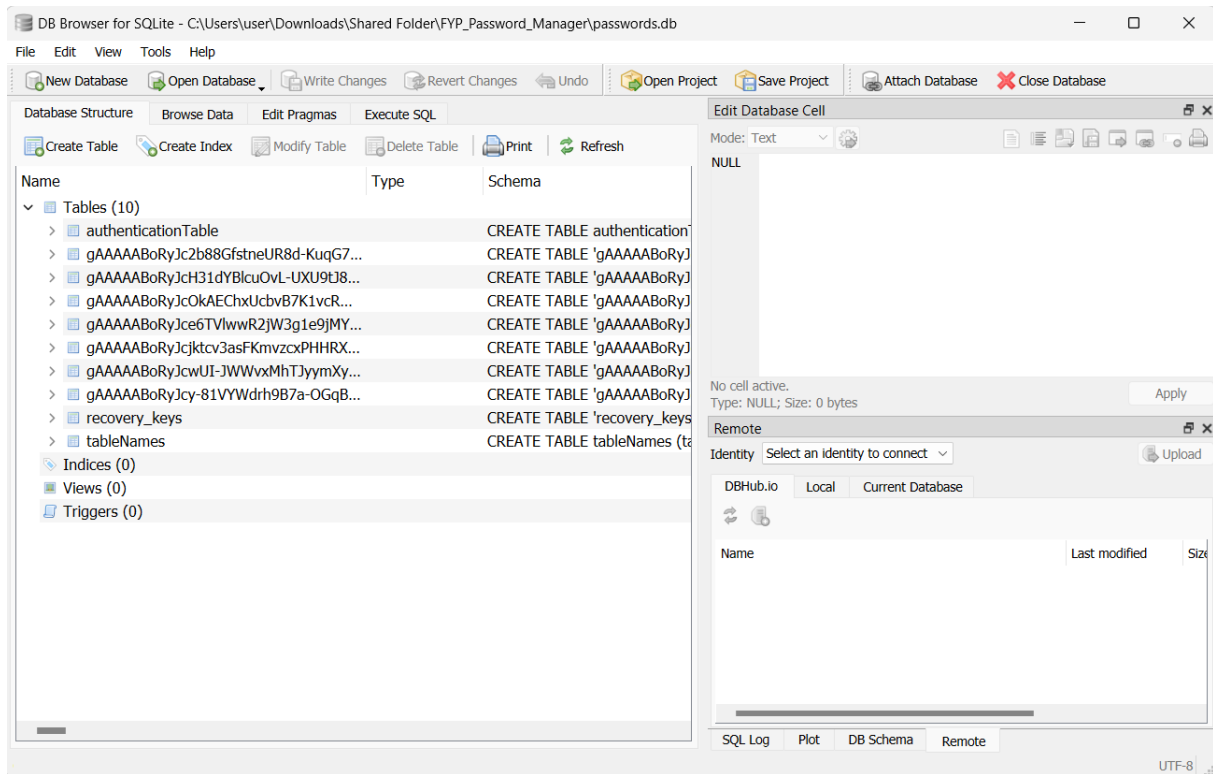


Figure 4 3 “passwords.db” File in DB Browser for SQLite

Note. This figure shows the “passwords.db” file opened in DB Browser for SQLite, a graphical tool for inspecting and managing encrypted SQLite databases.

Figure 4.3 shows the “passwords.db” file opened and viewed in DB Browser for SQLite¹¹, a graphical interface for inspecting and managing SQLite databases. The database stores encrypted passwords, user settings, and other related data. The system uses SQLite as the database to store all password entries and configuration data. SQLite was chosen for its lightweight nature, as it does not require an external server, simplifying the setup and management. The database file is named “passwords.db”, and it is created automatically during the first launch of the system. To enhance security, the database is encrypted using the `pysqlitecipher` library with SHA-512, which safeguards the contents and can only be accessed

¹¹ DB Browser for SQLite. *sqlitebrowser.org*. Available at: <https://sqlitebrowser.org/>

using pysqlitecipher specific commands instead of SQLite3. Detailed instructions on using pysqlitecipher can be found at this tutorial¹². Here are the steps required to set up DB Browser and pysqlitecipher for developing the Password Management System:

1. Download and install DB Browser for SQLite:

- Visit the official website: <https://sqlitebrowser.org/>
- Download the installer for your operating system and follow the installation instructions.
- This tool allows users to easily view, edit, and manage SQLite databases, including the “passwords.db” file.

2. Install pysqlitecipher:

- Instead of using the built-in sqlite3 module, the system uses pysqlitecipher for database encryption. Install it using pip in VS Code terminal by running the following command:

```
pip install pysqlitecipher==0.11
```

3. Access the encrypted database:

- After installing pysqlitecipher, you can use the following commands to initialize and interact with the encrypted database:

¹² Encrypting SQLite Database in Python Using pysqlitecipher Module. *Medium*. Available at: <https://medium.com/@harshnative/encrypting-sqlite-database-in-python-using-pysqlitecipher-module-23b80129fda0>

Step 1 — Initialize object:

```
from pysqlitecipher import sqlitewrapper

# Define schema for the new table
colList = [
    ["platform", "TEXT"],
    ["username", "TEXT"],
    ["password", "TEXT"]
]

# Create the table in the database
obj.createTable(tableName="password_entries", colList=colList, makeSecure=True,
commit=True)
```

Step 2 — Create Table:

To create a table in the encrypted database, you can use the `createTable()` method:

- **Parameters:**
 - **tableName:** The name of the table you want to create (e.g., "password_entries").
 - **colList:** The schema for the table in the form of a list of column names and their corresponding data types.
 - Example: ["platform", "TEXT"], ["username", "TEXT"], etc.
 - **makeSecure:** Set to "True" if you want to encrypt the data stored in this table.
 - **commit:** Set to "True" to commit the changes to the database.

Step 3 — Use encrypted database:

Once the table is created, you can continue using the database by performing other database operations, such as inserting, updating, and retrieving encrypted data, all while ensuring the data is protected by encryption.

For more detailed information on how to use `pysqlitecipher`, refer to this tutorial¹². Once these steps are completed, your system will be ready to use an encrypted SQLite database for securely storing and managing passwords.

4.2.4 Library Installation

Several essential Python libraries were used to implement the features of the system. These libraries include tools for encryption, clipboard management, GUI enhancements, and notifications. Below are the key libraries used:

- **cryptography** – for AES encryptions and PBKDF2 key derivation function
- **pyotp** – for one-time password (OTP) generation
- **pyperclip** – for managing clipboard operations
- **Pillow** – for handling image files
- **qrcode** – for generating QR codes
- **windows_toasts** – for displaying desktop alert notifications
- **pywin32** – for accessing Windows-specific features (e.g., clipboard, registry)

These libraries were installed via the VS Code terminal using the following command:

```
pip install library_name
```

4.2.5 Coding

The core functionality of the system revolves around encrypting and securely storing passwords, as well as providing an intuitive graphical user interface (GUI). Below are the key processes involved, along with corresponding code snippets for each critical function:

4.2.5.1 Obfuscating Database Password

The database password for the encrypted SQLite database is embedded and obfuscated in the program to ensure it's not easily accessible.

```
def transform_key(seed: int) -> str:
    """Generate a dynamic key using mathematical transformations"""
    key = []
    # Multi-step transformation using bitwise operations
    a = (seed ^ 0xDEADBEEF) + 0x2F7A83C
    b = (a << 3) | (a >> 29)
    c = b ^ 0x8C9A6B5
    d = (c * 0x2D) & 0xFFFFFFFF

    # Convert to characters using selective bit masking
    for i in range(12):
        val = d >> (i * 3) & 0x7F
        if 33 <= val <= 126: # Printable ASCII range
            key.append(chr(val))
        else:
            key.append(chr((val % 93) + 33))

    return ''.join(key)

def decode_password(encoded: bytes, key: str) -> str:
    """XOR decoding with key rotation"""
    return bytes([b ^ ord(key[i % len(key)]) for i, b in enumerate(encoded)]).decode()

def get_db_password() -> str:
    """Retrieve password through multiple obfuscation layers"""
    # Layer 1: Encoded byte sequence (appears as random bytes)
    encoded = b'\x12\x45\x28\x7C\x53\x69\x3F\x1A\x6E\x44\x22\x57\x31\x4A'

    # Layer 2: Dynamic key generation from mathematical seed
    key_seed = 0x6F2C8A1 # Appears as a random constant
    dynamic_key = transform_key(key_seed)

    # Layer 3: Multiple decoding passes
    temp = decode_password(encoded, dynamic_key)
    final = decode_password(temp.encode(), dynamic_key[::-1])

    # Layer 4: Final transformation
    return final.swapcase().translate(str.maketrans('s13', '$%3'))
```

Figure 4.4 Code Snippet for Obfuscation of the Database Password

Note. This figure shows the code snippet for obfuscating the database password, using a dynamically generated key and XOR decoding to secure the password.

Figure 4.4 shows the code snippet for the obfuscation of the database password. The password for the encrypted SQLite database is embedded in the program and obfuscated to make it difficult to access. The process involves using a dynamic key generation technique

where a seed value (`key_seed`) is transformed through a series of mathematical operations in the `transform_key()` function, using bitwise XOR, shifts, and other transformations. The generated key is then used in the `decode_password()` function to XOR decode the encoded password. The `get_db_password()` function retrieves the database password after multiple decoding passes, ensuring the password is securely obfuscated. To customize this, you can generate your own random byte sequence for the “encoded” variable by using tools like Browserling's Random Bytes¹³, then represent it in the format “\x..\x..” For the “key_seed” variable, you can convert any text or string into hexadecimal using the ASCII to Hex Converter¹⁴ and input it in the format “0x...”. to generate a unique key. This method ensures the password is obfuscated and not easily retrievable without the correct key and transformations.

4.2.5.2 Master Password Setup and Verification

During user login, the system prompts for a master password, which is securely stored in the database using PBKDF2 key derivation. The derived key is then used to encrypt and decrypt password entries with AES encryption (AES-128/192/256).

¹³ Random Bytes Generator. *Browserling*. Available at: <https://www.browserling.com/tools/random-bytes>

¹⁴ ASCII to Hex Converter. *RapidTables*. Available at: <https://www.rapidtables.com/convert/number/ascii-to-hex.html>

```

# Function to derive key from master password using PBKDF2
def derive_key(master_password, salt):
    """Generate a 256-bit key from the master password using PBKDF2."""
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32, # 256-bit key
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    return kdf.derive(master_password.encode())

def store_master_password(master_password):
    global sqlite_obj

    """Hash and store the master password using sqlitewrapper"""
    # Generate salt and derive key
    salt = os.urandom(16)
    key = derive_key(master_password, salt)

    # Store salt + key together
    hashed_password = base64.b64encode(salt + key).decode()

    # Delete any existing master passwords
    # First get all IDs in the table
    result = sqlite_obj.getDataFromTable("master_password", raiseConversionError=True, omitID=False)

    if result[1]: # If there are rows in the table
        # Delete each row by ID
        for row in result[1]: # result[1] contains the actual data
            id_value = int(row[0])
            sqlite_obj.deleteDataInTable("master_password", id_value, commit=True, updateId=True)

    # Insert the new master password
    sqlite_obj.insertIntoTable("master_password", [hashed_password], commit=True)

```

Figure 4.5 Master Password Setup and Key Derivation

Note. This figure demonstrates how the master password is used to generate a 256-bit key with PBKDF2 and a 16-byte salt.

Figure 4.5 shows the code snippet for setting up the master password and deriving a key using the PBKDF2 algorithm. The process begins when the user sets the master password, which is used to generate a salt and derive a key using PBKDF2. Specifically, the salt is 16 bytes long (128 bits), and a 256-bit key is generated. The PBKDF2 key derivation function is configured to run for 100,000 iterations to increase computational complexity, making brute-force attacks more difficult. The derived key and salt are then base64-encoded and securely stored in the database. The function `store_master_password()` manages this process by storing

the salt and derived key (not the master password itself) to ensure the security of the system. This derived key is used for encrypting and decrypting password entries.

```
def verify_master_password(entered_password):
    """Verify if the entered master password is correct and return the key."""
    global sqlite_obj

    # Fetch the hashed password from the "master_password" table
    result = sqlite_obj.getDataFromTable("master_password", raiseConversionError=True, omitID=False)

    if not result[1]: # No master password set
        return None

    stored_hashed_password = result[1][0][1] # The first row (0), second column (1) is the hashed password

    # Decode the base64-encoded hashed password
    try:
        stored_hashed_password = base64.b64decode(stored_hashed_password)
    except Exception as e:
        print(f"Error decoding hashed password: {e}")
        return None

    # Extract the salt and stored key
    salt = stored_hashed_password[:16] # Extract the salt
    stored_key = stored_hashed_password[16:] # Extract the stored key

    # Derive the key from the entered password
    entered_key = derive_key(entered_password, salt)

    if entered_key == stored_key:
        return entered_key # Return the actual key if the password is correct
    return None # Return None if the password is incorrect
```

Figure 4.6 Verifying Master Password during Login

Note. This figure illustrates how the `verify_master_password()` function verifies the entered password by comparing the derived key with the stored key using PBKDF2.

Figure 4.6 shows the code snippet for verifying the master password during login. When the user enters their master password, the system retrieves the stored hashed password from the database. The salt is extracted from the stored value, and the entered password is used to derive a new key using the same PBKDF2 process. If the derived key matches the stored key, the user is authenticated, and the derived key is returned for further use in encryption and decryption. The function `verify_master_password()` handles this verification process.

4.2.5.3 Password Encryption and Decryption

The key derived from the master password is used for AES encryption and decryption of password entries to ensure secure storage.

```
# Encrypt with AES based on selected AES bit size (128, 192, 256)
def encrypt_things(plain_text, key, aes_bits):
    iv = os.urandom(12) # Generate a random 12-byte IV for GCM
    if aes_bits == 128:
        cipher = Cipher(algorithms.AES(key[:16]), modes.GCM(iv), backend=default_backend()) # 128-bit AES
    elif aes_bits == 192:
        cipher = Cipher(algorithms.AES(key[:24]), modes.GCM(iv), backend=default_backend()) # 192-bit AES
    else: # AES-256
        cipher = Cipher(algorithms.AES(key), modes.GCM(iv), backend=default_backend()) # 256-bit AES

    encryptor = cipher.encryptor()
    encrypted_password = encryptor.update(plain_text.encode()) + encryptor.finalize()
    return base64.b64encode(iv + encryptor.tag + encrypted_password).decode('utf-8')

# Decrypt with AES based on selected AES bit size (128, 192, 256)
def decrypt_things(encrypted_text, key, aes_bits):
    try:
        encrypted_data = base64.b64decode(encrypted_text)
        iv = encrypted_data[:12] # First 12 bytes are the IV
        tag = encrypted_data[12:28] # Next 16 bytes are the GCM tag
        ciphertext = encrypted_data[28:] # Remaining bytes are the ciphertext

        if aes_bits == 128:
            cipher = Cipher(algorithms.AES(key[:16]), modes.GCM(iv, tag), backend=default_backend()) # 128-bit AES
        elif aes_bits == 192:
            cipher = Cipher(algorithms.AES(key[:24]), modes.GCM(iv, tag), backend=default_backend()) # 192-bit AES
        else: # AES-256
            cipher = Cipher(algorithms.AES(key), modes.GCM(iv, tag), backend=default_backend()) # 256-bit AES

        decryptor = cipher.decryptor()
        decrypted_password = decryptor.update(ciphertext) + decryptor.finalize()
        return decrypted_password.decode('utf-8')
    except Exception as e:
        return None # Return None in case of decryption error
```

Figure 4.7 Password Encryption and Decryption Using AES

Note. This figure depicts how the `encrypt_things()` function encrypts the password using AES in GCM mode and the derived key, while `decrypt_things()` reverses the process for decryption.

Figure 4.7 shows the process of password encryption and decryption using AES encryption. The function `encrypt_things()` encrypts the password using AES in GCM mode, based on the selected AES bit size (128, 192, or 256 bits). It generates a random 12-byte initialization vector (IV) and uses the key derived from the master password. The encrypted password is then combined with the IV and the authentication tag, and the result is base64-encoded. To decrypt the password, the `decrypt_things()` function is used, which reverses the

encryption process by extracting the IV, tag, and ciphertext, then decrypting it with the AES key.

4.2.5.4 Recovery Keys

Recovery keys are generated during account setup and stored securely. These keys are used to verify the user's identity if they forget their master password during login.

```
def generate_recovery_keys():
    """Generate 10 recovery keys and store their hashes in the database."""
    global password_verified
    # Only require master password once for backup settings
    if not password_verified:
        if not require_master_password():
            return
        password_verified = True # Set the flag after verifying the password

    # Generate 10 new recovery keys
    new_keys = [''.join(random.choices(string.ascii_uppercase + string.digits, k=8)) for _ in range(10)]
    hashed_recovery_keys = [hashlib.sha256(key.encode()).hexdigest() for key in new_keys]
    try:
        # Fetch existing recovery keys
        result = sqlite_obj.getDataFromTable("recovery_keys", raiseConversionError=True, omitID=False)

        if result[1]: # Check if there are existing keys
            for idx, row in enumerate(result[1]):
                row_id = int(row[0]) # Get the ID of the current row
                # Update the hashed_key in the table for each row
                sqlite_obj.updateInTable(
                    "recovery_keys",
                    row_id,
                    "hashed_key",
                    hashed_recovery_keys[idx], # Update with the corresponding new hashed key
                    commit=True,
                    raiseError=True
                )
    except Exception as e:
        print(f"Failed to save recovery keys: {e}")
        messagebox.showerror("Error", f"Failed to save recovery keys: {e}")
    # Display the new keys to the user
    for widget in keys_frame.winfo_children():
        widget.destroy()

    all_keys = ' '.join(new_keys)
    key_grid = tk.Frame(keys_frame, bg="white")
    key_grid.pack(fill=tk.BOTH, expand=True)
    # Display keys in a grid format
    for i in range(0, len(new_keys), 2):
        row_frame = tk.Frame(key_grid, bg="white")
        row_frame.pack(fill=tk.X, pady=2)
        tk.Label(row_frame, text=new_keys[i], font=("Helvetica", 11),
            bg="white", width=12, anchor="w").pack(side=tk.LEFT, padx=10)
        if i + 1 < len(new_keys):
            tk.Label(row_frame, text=new_keys[i + 1], font=("Helvetica", 11),
                bg="white", width=12, anchor="w").pack(side=tk.LEFT, padx=10)

    # Add a button to copy all keys
    copy_all_button = tk.Button(key_grid, text="Copy All Keys", font=("Helvetica", 11),
        command=lambda: copy_value(all_keys, root))
    copy_all_button.pack(pady=10)
    messagebox.showinfo("Recovery Keys Generated", "New recovery keys have been successfully generated.")
```

Figure 4.8 Generating Recovery Keys

Note. This figure shows the `generate_recovery_keys()` function, which creates 10 random recovery keys, hashes them with SHA-256, and stores the hashed keys in the database.

Figure 4.8 shows the code snippet for generating 10 recovery keys during account setup. The function `generate_recovery_keys()` creates random recovery keys, hashes them using SHA-256, and stores the hashed keys in the database. If recovery keys already exist, they are updated with the new hashes. The generated keys are displayed to the user, and a button is provided to copy all keys.

```
# Get recovery keys from the database
def get_recovery_keys():
    global sqlite_obj
    """Retrieve the hashed recovery keys from the database."""
    try:
        # Fetch hashed recovery keys from the database using sqlite_obj
        result = sqlite_obj.getDataFromTable(
            "recovery_keys",
            raiseConversionError=True,
            omitID=False
        )

        # Extract the hashed recovery keys from the result
        if result[1]:
            hashed_recovery_keys = [row[1] for row in result[1]]
        else:
            hashed_recovery_keys = []
        return hashed_recovery_keys

    except Exception as e:
        print(f"Error fetching recovery keys: {e}")
        return []

# Verify recovery key
def verify_recovery_key(entered_key, hashed_recovery_keys):
    """Check if the entered recovery key is valid."""
    entered_key_hash = hashlib.sha256(entered_key.encode()).hexdigest()
    return entered_key_hash in hashed_recovery_keys
```

Figure 4.9 Verifying Recovery Key During Login

Note. This figure shows how the `get_recovery_keys()` function retrieves hashed recovery keys and `verify_recovery_key()` checks if the entered key matches the stored key to proceed with login.

Figure 4.9 shows the code snippet for verifying a recovery key during login. The `get_recovery_keys()` function retrieves the stored hashed recovery keys from the database and

verify_recovery_key() compares the entered recovery key with the stored hashed keys. If a match is found, the login process proceeds, allowing the user to access the system.

4.2.5.5 Multi-Factor Authentication (MFA)

The system implements MFA using One Time Password (OTP). The OTP setup key is encrypted with the master password-derived key, and during login, it is verified to ensure the integrity of the MFA process.

```
def show_qr_and_otp():
    otp_secret = pyotp.random_base32()
    totp = pyotp.TOTP(otp_secret)
    qr_code = qrcode.make(totp.provisioning_uri("", issuer_name="Password Manager"))

    # Reduce the size of the QR code
    qr_code = qr_code.resize((200, 200), Image.LANCZOS)

    qr_code_img = ImageTk.PhotoImage(qr_code)

    # Place the setup key before the QR code
    tk.Label(settings_frame, text=f"Setup Key: {otp_secret}", font=("Helvetica", 10)).pack(pady=5)
    qr_label = tk.Label(settings_frame, image=qr_code_img)
    qr_label.image = qr_code_img
    qr_label.pack(pady=10)

    # Instructions for the user
    tk.Label(settings_frame, text="Use an authenticator app to scan the QR code or enter the setup key.",
            font=("Helvetica", 9), wraplength=500, justify="left").pack(pady=5)

    # OTP entry field
    tk.Label(settings_frame, text="Enter the OTP code from your authenticator app:",
            font=("Helvetica", 12)).pack(pady=10)
    otp_entry = tk.Entry(settings_frame, font=("Helvetica", 10), justify="center")
    otp_entry.pack(pady=10)
    otp_entry.focus_set() # Set focus to the entry field

def verify_and_save():
    entered_otp = otp_entry.get()
    totp = pyotp.TOTP(otp_secret)
    if totp.verify(entered_otp):
        encrypted_otp_secret = encrypt_things(otp_secret, key, 256)
        save_settings('mfa', True, show_mfa_settings)
        save_settings('otp_secret', encrypted_otp_secret, show_mfa_settings)
        messagebox.showinfo("Success", "MFA settings saved and verified.")
    else:
        messagebox.showerror("Error", "OTP verification failed. MFA settings not saved.")

verify_button = tk.Button(settings_frame, text="Verify and Save",
        font=("Helvetica", 9), command=verify_and_save)
verify_button.pack(pady=10)

# Bind the Enter key to the verify_and_save function
otp_entry.bind('<Return>', lambda event: verify_and_save())
```

Figure 4.10 Generating OTP and Displaying QR Code

Note. This figure illustrates how the `show_qr_and_otp()` function generates a secret key, creates a QR code for MFA setup, and encrypts the OTP setup key using the key derived from the master password as shown in Figure 4.5.

Figure 4.10 presents the code snippet for generating a One-Time Password (OTP) when the MFA option is enabled in the Settings Page of the system. The function `show_qr_and_otp()` generates a secret key using `pyotp`, creates a QR code for the user to scan with an authenticator app, and displays the setup key along with the QR code. The OTP setup key is encrypted using the key derived from the master password as shown in Figure 4.5. The user is then prompted to enter the generated OTP, which is verified before saving the MFA settings securely.

```
def verify_otp(encrypted_otp_secret):
    otp_secret = decrypt_things(encrypted_otp_secret, key, 256) # Decrypt the OTP secret

    def submit_otp():
        entered_otp = otp_entry.get()
        totp = pyotp.TOTP(otp_secret)
        if totp.verify(entered_otp):
            otp_window.result = True
            otp_window.destroy()
        else:
            messagebox.showerror("Invalid OTP", "Incorrect OTP code. Try again.")

    def on_enter(event):
        submit_otp()

    def center_window(window, width, height):
        screen_width = window.winfo_screenwidth()
        screen_height = window.winfo_screenheight()
        position_top = int(screen_height / 2 - height / 2)
        position_right = int(screen_width / 2 - width / 2)
        window.geometry(f'{width}x{height}+{position_right}+{position_top}')

    otp_window = tk.Tk()
    otp_window.title("OTP Verification")
    otp_window.configure(bg="#F0F0F0")
    otp_window.result = False # Default to False unless OTP is verified

    otp_window.iconbitmap(default="./Images/logo.ico")

    center_window(otp_window, 400, 250)

    tk.Label(otp_window, text="Enter the OTP code:", bg="#F0F0F0", font=("Arial", 12)).pack(pady=10)

    otp_entry = tk.Entry(otp_window, font=("Arial", 14), justify="center")
    otp_entry.pack(pady=10)

    submit_button = tk.Button(otp_window, text="Verify", command=submit_otp, font=("Arial", 12), bg="#4CAF50", fg="white")
    submit_button.pack(pady=10)

    otp_window.bind('<Return>', on_enter)

    # Ensure focus after window appears
    otp_window.lift()
    otp_window.focus_force()
    otp_window.after(100, lambda: otp_entry.focus())

    otp_window.wait_window() # Wait until OTP window closes
    return otp_window.result
```

Figure 4.11 Verifying OTP During Login

Note. This figure demonstrates how the `verify_otp()` function decrypts the stored OTP secret and verifies the entered OTP using `pyotp`.

Figure 4.11 shows the code snippet for verifying the OTP during login. The function `verify_otp()` decrypts the stored OTP secret, which was encrypted using the key derived from the master password, then compares the entered OTP with the generated OTP using `pyotp`. If the OTP is correct, the user is authenticated; otherwise, an error message is displayed, prompting the user to try again.

4.2.5.6 Password Generation

The system includes a built-in password generator, which creates secure passwords based on user-defined criteria (e.g., length, complexity). The code snippet for generating a secure password can be found at Appendix B below. It is implemented using the `generate_password()` function, which creates a password based on user-defined criteria such as length and character complexity. The `generate_now()` function retrieves the password criteria from the database, including options for including uppercase letters, lowercase letters, digits, special characters, and other specific symbols. It builds a character set by including the appropriate characters based on the user's criteria (e.g., `string.ascii_uppercase`, `string.digits`, `string.punctuation`, and more). The password is then generated by randomly selecting characters from this set. If the generated password meets the strength criteria (labeled as "Strong" by the `check_password_strength()` function), it is inserted into the user interface; otherwise, the system prompts the user to adjust the criteria.

4.2.5.7 Password Breach Check

Using the Pwned Passwords API¹, the system checks if the password has been exposed in any known data breaches, helping users choose a safer password.

```

def check_pwned_password(password: str):
    if not password:
        return "⚠️ No password entered.", None, True

    hashed_password = hashlib.sha1(password.encode()).hexdigest().upper()
    prefix = hashed_password[:5]
    api_url = f'https://api.pwnedpasswords.com/range/{prefix}'

    try:
        response = requests.get(api_url, timeout=5)
        if response.status_code == 200:
            hashes = response.text.split('\r\n')
            for line in hashes:
                hash_suffix, count = line.split(':')
                if hash_suffix == hashed_password[5:]:
                    return f"⚠️ This password has been seen {count} times in breaches. You should change it.", int(count), False
            return "✅ Good news – this password has not been found in any known data breaches.", None, False
        else:
            return f"⚠️ API Error: Status code {response.status_code}", None, True
    except requests.exceptions.RequestException as e:
        return f"⚠️ Network error: {str(e)}", None, True
    except Exception as e:
        return f"⚠️ Unexpected error: {str(e)}", None, True

```

Figure 4.12 Password Breach Check

Note. This figure demonstrates how the `check_pwned_password()` function checks if a password has been exposed in data breaches using the Pwned Passwords API¹, alerting users if the password is compromised.

Figure 4.12 demonstrates how the `check_pwned_password()` function checks if a password has been exposed in known data breaches using the Pwned Passwords API¹. The function hashes the password with SHA-1, retrieves the hashed prefix from the API, and checks if the password's hash suffix matches any records. If a match is found, the user is alerted with the number of breaches; otherwise, a success message is displayed. This helps users choose safer passwords by checking them against public breach databases.

4.2.5.8 Password Strength Testing

The system evaluates password strength by simulating four types of attacks: password brute force, AES brute force, dictionary, and rainbow table attacks, estimating the time required to crack them and comparing it to the actual attack crack time, providing users with a transparent way to assess the strength of their password against real-world attacks.

4.2.5.8.1 Estimated Time to Crack for Four Attacks

The code snippet for estimating the time required to crack a password using four different attack methods—password brute force, AES brute force, dictionary attack, and rainbow table attack—can be found in Appendix C. The `estimate_crack_time()` function calculates the estimated cracking time for each method based on the password's entropy and the number of guesses per second (`guesses_per_sec`), which can either be user-defined or determined by running a test on the user's device to measure the guess rate, as demonstrated in Figure 4.13 below. The following outlines the logic for estimating the cracking time for each attack:

1. Password Brute Force:

- This method estimates the time by calculating the possible combinations of characters in the password based on the character set size and length. The cracking time is calculated as the total number of possible combinations divided by the guesses per second.

2. AES Brute Force:

- For AES brute force, the function calculates the time by taking $2^{\text{aes_bits}}$ (where `aes_bits` can be 128, 192, or 256 bits), which represents the number of possible AES keys, and divides it by the number of guesses per second.

3. Dictionary Attack:

- This method estimates the time by checking the password against a dictionary file. The function calculates the number of words in the dictionary and estimates the time required to check each word against the password.

- The default dictionary list used is sourced from the SecLists Project¹⁵, specifically from the /Passwords/ directory.
- If the password is found in the dictionary, the system will return a message with the estimated time to crack; otherwise, it will indicate that the password was not found.

4. Rainbow Table Attack:

- In the case of rainbow table attacks, the function utilizes precomputed hash values stored in rainbow tables. The estimated cracking time is determined by the password length, the number of threads, and the base time, which represents the average time derived from running the rcracki_mt program¹⁶ on passwords of lengths ranging from 1 to 7 on the laptop specified from Chapter 4.2.1: Hardware Specification, using the MD5 rainbow tables (mixalpha-numeric-symbol32-space#1-7 schema) from.
- These tables cover all character combinations with lengths from 1 to 7, including lowercase and uppercase letters (a-z, A-Z), digits (0-9), and special characters (e.g., !@#\$%^&*()_+=[\]{}|;:'",.< > ? /). Due to storage constraints on the laptop used, only the precomputed MD5 hash tables (mixalpha-numeric-symbol32-space#1-7) (86 GB) are used.
- The rcracki_mt program is stored in the Crack_Rainbow_Table directory within the system folder, while the rainbow table data resides in the Rainbow_Table directory.
- The function retrieves the base time for the corresponding password length from the database, adjusts it based on the number of threads, and calculates the time needed to

¹⁵ SecLists Project – Passwords Directory. *GitHub*. Available at: <https://github.com/danielmiessler/SecLists/tree/master/Passwords>

¹⁶ rcracki_mt. *GitHub*. Available at: https://github.com/foreni-packages/rcracki_mt

crack the password. If no estimate is available for the specified password length, an appropriate message is returned. In case of errors, a database error message is displayed.

4.2.5.8.2 Measuring Guess Rate on User's Device

```
def run_guess_rate_test():
    # Format the start message with timestamp
    start_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    message_queue.put(f"🚀 Starting guess rate test with random password generated: '{random_password}'")
    message_queue.put(f"🕒 Start time: {start_time}")
    message_queue.put(f"🧵 Using {thread_count} threads and threshold: {threshold:,}")

    chunk_size = len(alphabet) // thread_count
    chunks = [alphabet[i * chunk_size:(i + 1) * chunk_size] for i in range(thread_count - 1)]
    chunks.append(alphabet[(thread_count - 1) * chunk_size:])

    def brute_force(limit, chunk, pwd, stop_flag):
        for guesses, g in enumerate(itertools.product(chunk, repeat=len(pwd)), 1):
            if stop_flag and stop_flag.is_set():
                return
            if ''.join(g) == pwd:
                message_queue.put(f"🔑 Password cracked: '{''.join(g)}'")
                break
            if guesses >= limit:
                break

    start = time.time()
    threads = [
        threading.Thread(
            target=brute_force,
            args=(threshold // thread_count, chunk, random_password, test_stop_flag)
        ) for chunk in chunks
    ]
    for t in threads:
        t.start()
    for t in threads:
        t.join()

    if test_stop_flag.is_set():
        message_queue.put("🛑 Guess rate test stopped by user.")
        message_queue.put("")
    else:
        taken = time.time() - start
        guess_per_sec = round(threshold / taken)
        end_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Format the completion message
        message_queue.put(f"🕒 End time: {end_time}")
        message_queue.put(f"⌚ Time taken: {taken:.2f} seconds")
        message_queue.put(f"📊 New Guess/sec: {guess_per_sec:,}")
        message_queue.put("🏁 Guess rate test completed")
        message_queue.put("")
```

Figure 4.13 Measuring Guess Rate on User's Device

Note. This figure illustrates the `run_guess_rate_test()` function, which calculates the guess rate (guesses per second) on the user's device by simulating a brute force attack, and the result is used in Appendix C to estimate password cracking times for different attack methods.

Figure 4.13 shows the `run_guess_rate_test()` function, which measures the guess rate (guesses per second) on the user's device by simulating a password cracking attack using brute force. The function generates a random 12-character password using letters, digits, and

punctuation, then divides the character set into chunks based on the number of threads specified for parallel processing. Each thread handles a portion of the set and generates all possible combinations, stopping once the password is found or the guess limit is reached.

The time taken for the test is recorded, and the guess rate is calculated by dividing the total number of attempts by the time spent. The result is displayed to the user, showing how many guesses per second the system can handle. This guess rate can then be used in Appendix C to estimate the time required to crack passwords using different attack methods. This method allows users to test and customize their device's cracking speed for more accurate predictions of password vulnerabilities against real-world attacks.

4.2.5.8.3 Actual Password Brute Force Attack Time

```
def simulate_attack(method, threads, password, dict_path, rainbow_path, queue, start_button, stop_button, stop_flag):
    def brute_force_worker(start_chars, charset, target, found_event, attempt_counter, stop_flag, length_status_callback):
        for first in start_chars:
            for combo in itertools.product(charset, repeat=length - 1):
                if found_event.is_set() or stop_flag.is_set():
                    return
                attempt = first + ''.join(combo)
                attempt_counter[0] += 1
                if attempt == target:
                    found_event.set()
                    post(f"✅ Password found: {attempt}")
                    return

    start_time = time.time()
    attempt_counter = [0]

    if method == "Password Brute-Force":
        charset = string.ascii_letters + string.digits + string.punctuation
        post("🌀 Starting password brute-force with increasing lengths...")
        post(f"📏 Charset size: {len(charset)}")

        found_event = threading.Event()
        workers = []
        current_length = [0]
        length_lock = threading.Lock()

        def length_status_callback(length):
            with length_lock:
                if current_length[0] != length:
                    current_length[0] = length
                    post(f"🔄 Trying password length: {length}")

        for i in range(threads):
            start_chars = charset[i::threads]
            t = threading.Thread(
                target=brute_force_worker,
                args=(start_chars, charset, password, found_event, attempt_counter, stop_flag, length_status_callback),
                daemon=True
            )
            workers.append(t)
            t.start()

        for t in workers:
            t.join()

        if not found_event.is_set() and not stop_flag.is_set():
            post("❌ Password not found (unexpected)")
```

Figure 4.14 Actual Password Brute Force Attack Simulation

Note. This figure shows the `simulate_attack()` function, which performs a password brute force attack by progressively testing combinations from length 1 to the actual password length, using multiple threads to calculate the time taken to crack the password.

Figure 4.14 demonstrates the code snippet for performing a password brute force attack using the `simulate_attack()` function. The function simulates the attack by starting with combinations of characters for password lengths of 1 and continues increasing the length until it reaches the actual password length and successfully identifies the password. The attack

utilizes multiple threads for efficient execution and calculates the actual time taken to crack the password by testing all possible combinations.

4.2.5.8.4 Actual AES Brute Force Attack Time

```
elif method == "AES Brute-Force":
    try:
        aes_bits = int(aes_bit)
        key_length = aes_bits // 8
    except:
        post("❌ Invalid AES bit size")
        return

    charset = string.printable.strip()
    charset_size = len(charset)
    post(f"🔧 Starting AES-{aes_bits} brute-force with key length {key_length} bytes...")
    post(f"📏 Charset size: {charset_size}")
    post("⚠️ WARNING: This operation is computationally infeasible with current computer technology!")

    plaintext = "secret"
    target_key = password.encode().ljust(key_length, b'\0')[:key_length]
    iv = os.urandom(12)
    cipher = Cipher(algorithms.AES(target_key), modes.GCM(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext.encode()) + encryptor.finalize()
    tag = encryptor.tag
    target_ciphertext = base64.b64encode(iv + tag + ciphertext).decode()
    post(f"🎯 Target ciphertext: {target_ciphertext}")

    found_event = threading.Event()
    workers = []
    attempt_counter = [0] # reuse outer stop_flag - do NOT redefine

    def aes_brute_worker(start_chars, charset, key_length, found_event, attempt_counter, stop_flag):
        for first_char in start_chars:
            for tail in itertools.product(charset, repeat=key_length - 1):
                if found_event.is_set() or stop_flag.is_set():
                    return
                candidate_str = first_char + ''.join(tail)
                candidate_key = candidate_str.encode().ljust(key_length, b'\0')[:key_length]
                try:
                    cipher = Cipher(algorithms.AES(candidate_key), modes.GCM(iv), backend=default_backend())
                    encryptor = cipher.encryptor()
                    ct = encryptor.update(plaintext.encode()) + encryptor.finalize()
                    tag = encryptor.tag
                    test_ciphertext = base64.b64encode(iv + tag + ct).decode()
                    attempt_counter[0] += 1

                    if attempt_counter[0] % 100000 == 0:
                        post(f"🔄 Attempt {attempt_counter[0]}: Trying key '{candidate_str}' → {candidate_key.hex()}")

                    if test_ciphertext == target_ciphertext:
                        found_event.set()
                        post(f"✅ Key found: '{candidate_str}' (hex: {candidate_key.hex()})")
                        return
                except Exception:
                    pass

    for i in range(threads):
        start_chars = charset[i::threads]
        t = threading.Thread(
            target=aes_brute_worker,
            args=(start_chars, charset, key_length, found_event, attempt_counter, stop_flag),
            daemon=True
        )
        workers.append(t)
        t.start()

    for t in workers:
        t.join()

    if not found_event.is_set() and not stop_flag.is_set():
        post("❌ AES key not found in search space")
```

Figure 4.15 Actual AES Brute Force Attack Simulation

Note. This figure shows the AES brute force attack using the `simulate_attack()` function from Figure 4.14, employing multithreading to test various AES keys based on the selected bit size and displaying a

warning about the impracticality of brute-forcing AES encryption with current technology due to its computational intensity.

Figure 4.15 demonstrates the code snippet for performing an AES brute force attack on an encrypted password. The attack continues using the `simulate_attack()` function from Figure 4.14 and utilizes multithreading for efficiency. The function tests various AES keys based on the selected bit size (128, 192, or 256) and calculates the actual time required to break the encryption. Due to the computationally intensive nature of AES encryption, a warning will be displayed, indicating that brute-forcing AES encryption with current computer technology is virtually impossible because of the immense time it takes.

4.2.5.8.5 Actual Dictionary Attack Time

Appendix D illustrates the code snippet for performing a dictionary attack using the `simulate_attack()` function. The function attempts to crack the password by comparing it against entries in a dictionary file, as detailed in Dictionary Attack (Chapter 4.2.5.8.1). It calculates the actual time taken to find a match by iterating through the dictionary, which contains a list of common passwords. The function retrieves the dictionary from the specified path, tests each word, and determines if it matches the target password. If the password is found, the function reports the time taken to crack it. If the password is not found in the dictionary, the attack will automatically proceed to a password brute force attack.

4.2.5.8.6 Actual Rainbow Table Attack Time

Appendix E shows the code snippet for performing a rainbow table attack on a password hash using the `simulate_attack()` function. The function uses precomputed hashes from a rainbow table to attempt to crack the password and calculates the actual time taken to find a match, utilizing the `rcracki_mt` tool, as mentioned in Rainbow Table Attack (Chapter 4.2.5.8.1).

The function automatically generates the MD5 hash of the password entered by the user and executes the command `rcracki_mt -h {md5_hash} -t {threads} {rainbow_path}` to initiate the cracking process, utilizing multithreading for efficient execution.

4.2.5.9 AES Bit Evaluation

The system evaluates three AES bit options—128, 192, and 256—to help users select the most appropriate AES bit size for encrypting password entries while balancing performance and security. To facilitate this selection, users can perform a guess per second test on their system to assess its computational power for password cracking, as outlined in Chapter 4.2.5.8.2. The resulting guess rate is then used to estimate the cracking time for each AES bit size, allowing users to determine the best option for their device.

Appendix F presents performance metrics for each AES bit size, including encryption and decryption time, memory usage, and throughput. These metrics help users understand the impact of each AES bit size on system performance. Appendix G provides security metrics such as ciphertext entropy and estimated crack time, offering insights into how each AES bit size affects the strength of the encryption. Appendix H details how to evaluate and select the best AES bit for a user's device by scoring each bit size based on the performance and security metrics. This process ensures that users can make an informed decision to achieve an optimal balance between security and system efficiency.

4.3 The Developed System

This section describes the developed Password Management System, showcasing its key features and providing screenshots of the system's interface, as shown in Figures 4.16 to 4.35.

4.3.1 Login Screen

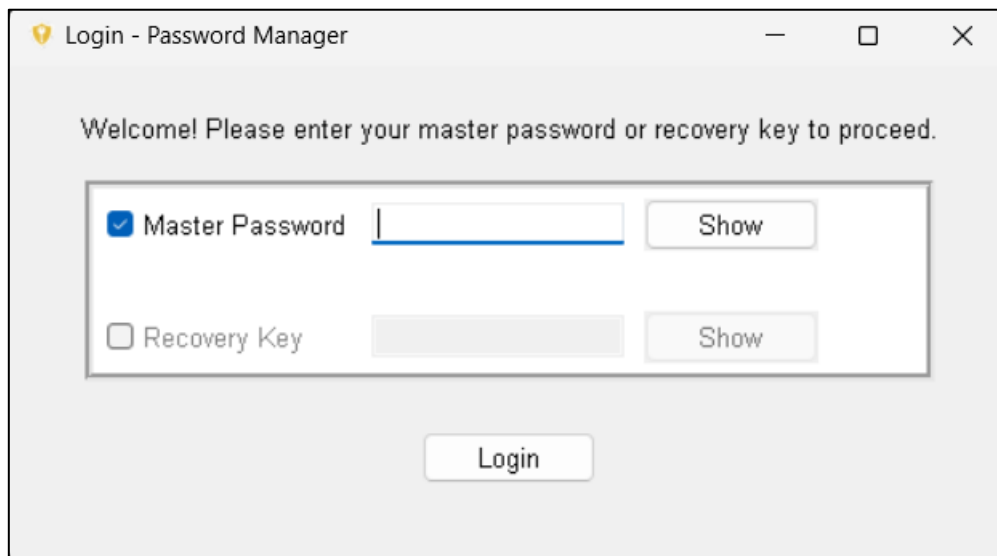


Figure 4.16 Login Screen

Note. This figure displays the login screen with two authentication options: entering the master password or using recovery keys for access.

The login screen of the system, as shown in Figure 4.16, provides two options for login: entering the master password or using recovery keys. This screen allows users to choose between verifying their identity with their master password or, if they have forgotten it, using one of the recovery keys. The option to use recovery keys provides an alternative authentication method, ensuring users can still access the system even if they cannot remember their master password.

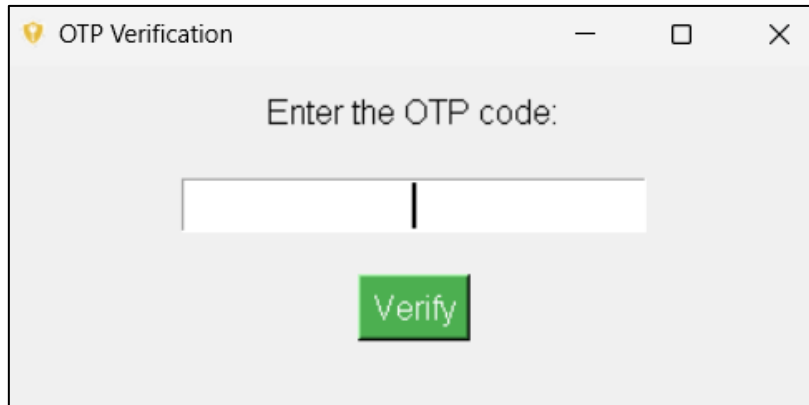


Figure 4.17 Login Screen

Note. This figure displays the OTP verification menu, which appears after a successful login with either the master password or recovery keys when Multi-Factor Authentication (MFA) is enabled, prompting users to enter a One-Time Password (OTP) to complete the login process.

Figure 4.17 shows the OTP verification menu that appears after successfully logging in with either the master password or recovery keys, provided that MFA is enabled in the system settings. This menu prompts users to enter OTP generated by their authenticator app to complete the login process. If MFA is turned on, this additional layer of security ensures that only authorized users can access the system, even if the master password or recovery key is compromised.

4.3.2 Main Menu

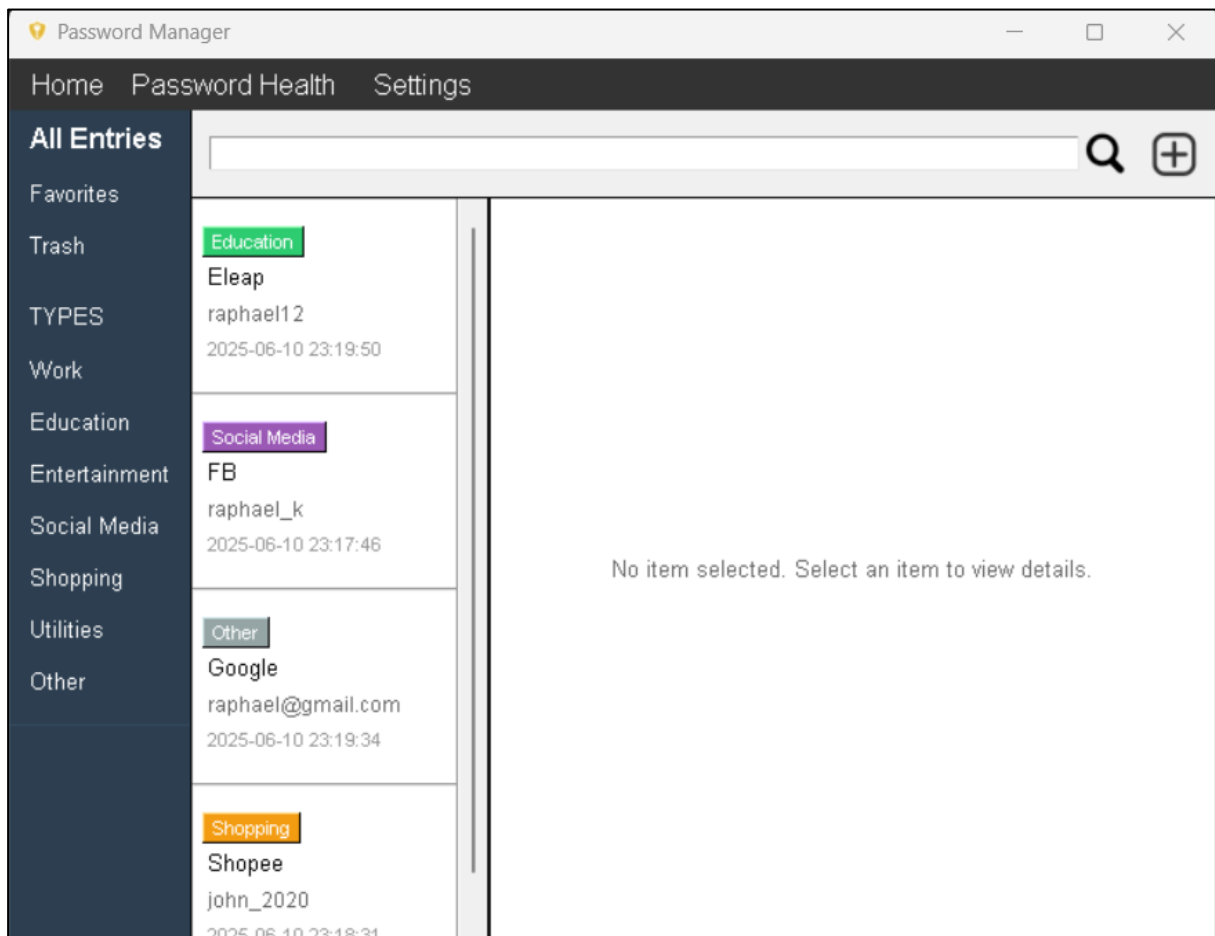


Figure 4.18 Main Menu

Note. This figure illustrates the main menu of the system, featuring a navigation bar for easy access to the home, password health, and settings menus, a sidebar to filter and edit password entries, and options to search and add new entries.

Figure 4.18 showcases the main menu of the system, which includes a top navigation bar with options for "Home," "Password Health," and "Settings." These options can be selected to navigate to their respective pages: the main menu, password health menu, and settings menu. On the left side of the screen, a sidebar allows users to filter and view existing password entries. Each entry can be clicked to display its details and enable editing. Above the password entries, a search bar helps users quickly locate specific entries, while a "+" button provides an option to add new password entries to the system.

4.3.3 Add/Edit Password Entries

The screenshot shows the 'Password Manager' application window. The top navigation bar includes 'Home', 'Password Health', and 'Settings'. A left sidebar lists categories: 'All Entries', 'Favorites', 'Trash', 'TYPES', 'Work', 'Education', 'Entertainment', 'Social Media', 'Shopping', 'Utilities', and 'Other'. The main content area is titled 'ADD NEW ENTRY' and contains the following fields and controls:

- Platform Name:** A text input field with a red asterisk indicating it is required.
- Label:** A dropdown menu with 'Work' selected and a red asterisk.
- Username:** A text input field with a red asterisk.
- Password:** A text input field with a red asterisk. To its right are three icons: an eye (toggle visibility), a checkmark (check for breaches), and a circular arrow (generate password).
- Confirm Password:** A text input field with a red asterisk.
- AES Bit:** A dropdown menu with '256' selected and a red asterisk. To its right is a blue button labeled 'Pick for Me'.

At the bottom of the form are two buttons: a red 'Cancel' button and a green 'Save' button.

Figure 4.19 Add New Password Entry Form

Note. This figure shows the form for adding a new password entry, allowing users to input platform details, select AES encryption bits, manage password visibility, check for breaches, generate passwords, and configure additional settings.

Figure 4.19 displays the form for adding a new password entry, where users can input details such as the platform name, username, password, strength indicator, notes, URL, and select the desired AES encryption bits. The form provides options for password management, including showing or hiding the password, checking for breaches, generating passwords, and copying the password. Additionally, the form allows users to configure password settings such as length, character inclusion, and more. Users can also enable a "Master Password Reprompt" option for added security and mark the entry as a favorite.

The screenshot displays the 'Password Manager' application window. The top navigation bar includes 'Home', 'Password Health', and 'Settings'. The left sidebar lists various categories: 'All Entries', 'Favorites', 'Trash', 'TYPES', 'Work', 'Education', 'Entertainment', 'Social Media', 'Shopping', 'Utilities', and 'Other'. The main content area is titled 'EDIT ENTRY' and contains the following fields and controls:

- Platform Name:** A text field containing 'Eleap' with a copy icon to its right.
- Label:** A dropdown menu currently set to 'Education'.
- Username:** A text field containing 'raphael12' with a copy icon to its right.
- Password:** A masked text field (represented by asterisks) with icons for showing/hiding the password, checking for breaches, generating a new password, and copying the password.
- Confirm Password:** A masked text field.
- AES Bit:** A dropdown menu set to '256' and a 'Pick for Me' button.

At the bottom of the form are three buttons: 'Delete' (orange), 'Cancel' (red), and 'Save' (green).

Figure 4.20 Edit Password Entry Form

Note. This figure presents the form for editing an existing password entry, enabling users to modify platform details, adjust password settings, check for breaches, and configure AES encryption while securely storing the updated information.

Figure 4.20 shows the form for editing an existing password entry, where users can modify details such as the platform name, username, password, strength indicator, notes, URL, and AES encryption bits. The form includes options for adjusting password management settings like showing/hiding the password, checking for breaches, generating new passwords, and copying the password. It also supports the master password reprompt option and allows users to mark the entry as a favorite, ensuring updated information is securely stored in the system.

4.3.4 Password Generation

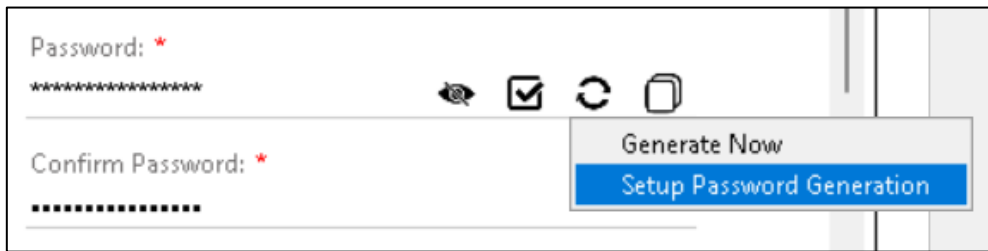


Figure 4.21 Context Menu for Generate Password Button

Note. This figure displays the context menu for the "Generate Password" button, offering options to instantly generate a secure password or customize the password generation criteria such as length, character types, and complexity.

The context menu for the "Generate Password" button, represented by the reload icon, as illustrated in Figure 4.21, provides two options: "Generate Now" and "Setup Password Generation". The "Generate Now" option instantly generates a secure password based on the set criteria, while the "Setup Password Generation" option opens a configuration menu where users can customize the password generation process by specifying parameters such as length, the inclusion of digits, uppercase and lowercase letters, special characters, and other customizable options for password complexity.



Figure 4.22 Password Generation Configuration Menu

Note. This figure illustrates the password generation configuration menu, where users can customize password parameters, including length and character types, and save, discard, or reset their settings as needed.

Figure 4.22 displays the password generation configuration menu, allowing users to specify parameters for generating secure passwords, including length, inclusion of digits, uppercase and lowercase letters, special characters, and other customizable options to adjust password complexity. Users can press "OK" to save their selected criteria, "Cancel" to discard any changes made during the customization process, or "Reset to Defaults" to restore the password generation settings to their default values, ensuring flexibility and control over the password generation process.

4.3.5 Password Breach Check

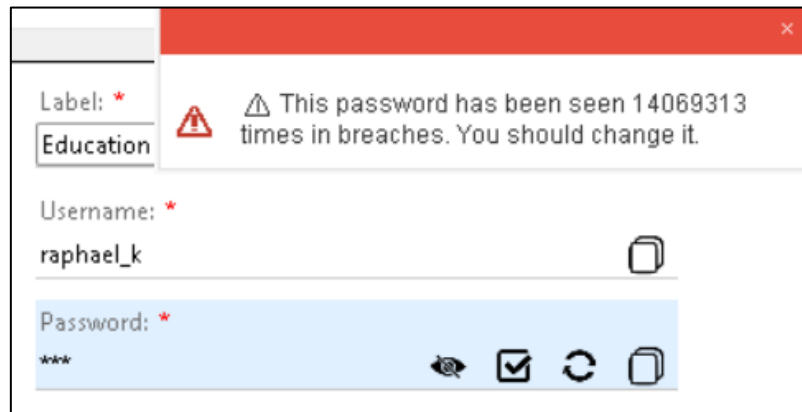


Figure 4.23 Password Breached

Note. This figure displays the result when the password, checked using the breach check button is found to have been exposed in known data breaches via the Pwned Passwords API¹.

Figure 4.23 shows the result when the password, checked using the breach check button represented by a tick icon in the password field, is found to be breached by the Pwned Passwords API¹. The system hashes the password, sends the hash to the API, and checks if it appears in any known data breaches. If the password is found in a breach, a warning is displayed to the user, indicating the password's vulnerability.

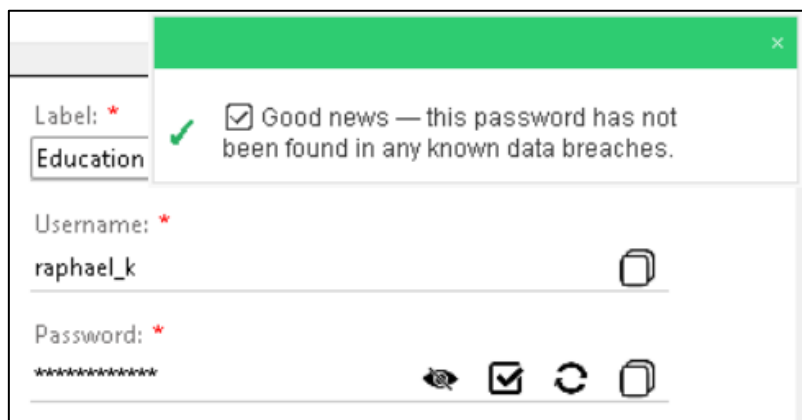


Figure 4.24 Password Not Breached

Note. This figure illustrates the result when the password, checked using the breach check button is verified as safe, indicating it has not been found in any known data breaches.

Figure 4.24 displays the result when the password, checked using the breach check button, is not found in any known data breaches. The system confirms the password's safety by cross-referencing it with the Pwned Passwords API¹. If no breach is found, the password is deemed secure, and the user is informed that the password has not been exposed in any known breaches.

4.3.6 Password Strength Tester Menu

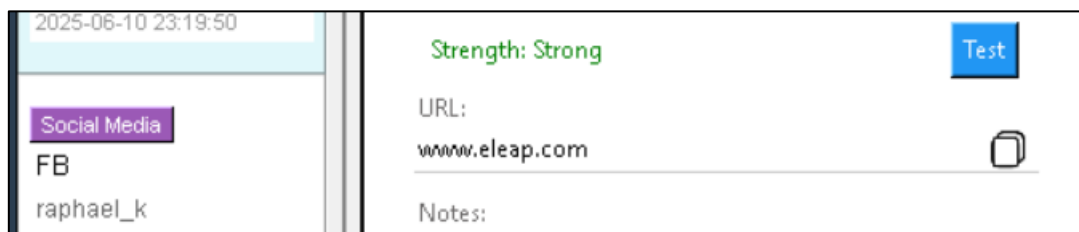


Figure 4.25 Password Strength Test Button

Note. This figure displays the "Test" button next to the password strength indicator, which opens the password strength tester menu when clicked.

Figure 4.25 shows the "Test" button located next to the password strength indicator. When clicked, this button opens the password strength tester menu, allowing users to evaluate their password's strength against four types of attacks: password brute force, AES brute force, dictionary attack, and rainbow table attack.

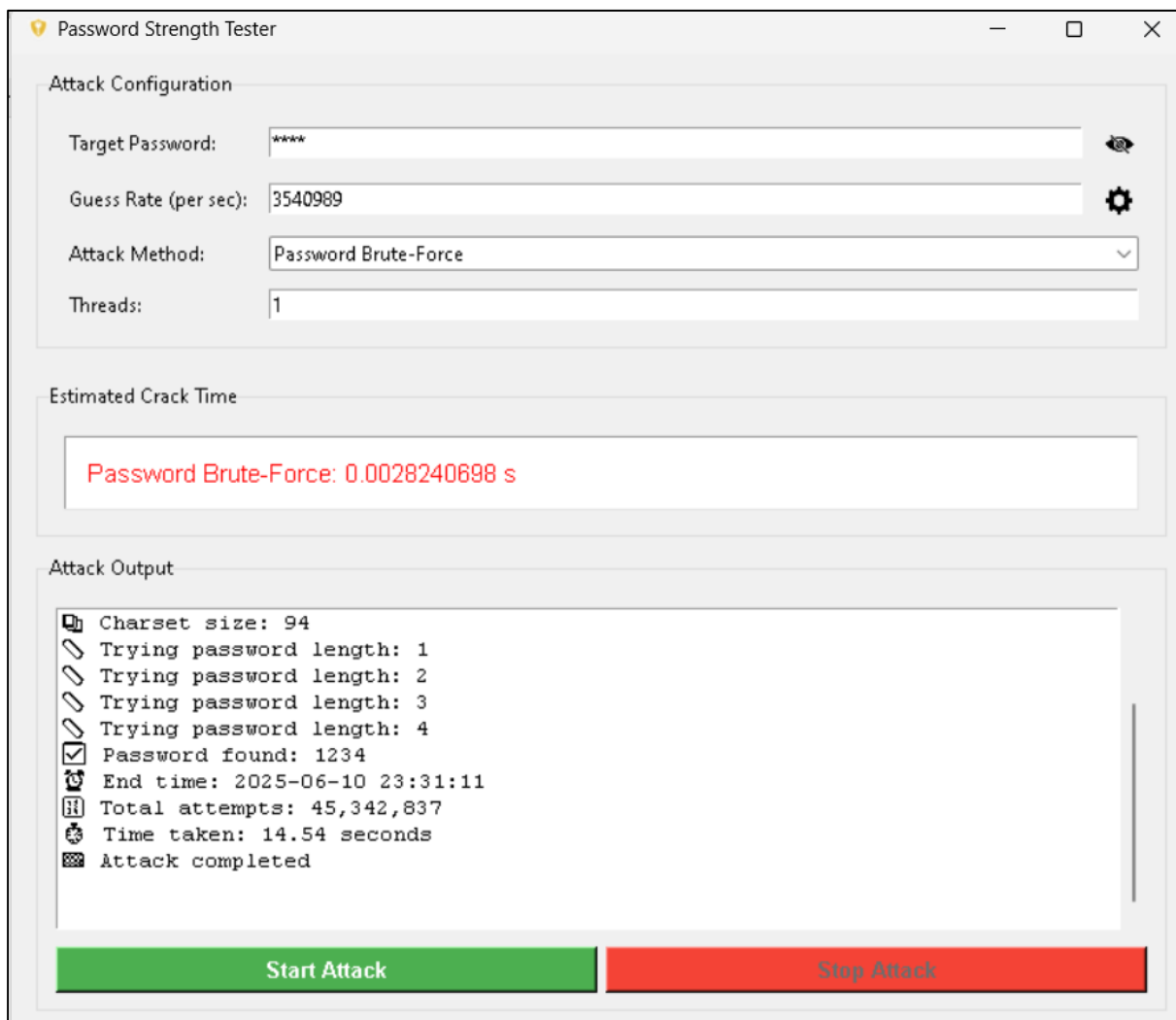


Figure 4.26 Password Strength Tester Menu

Note. This figure illustrates the result when the password, checked using the breach check button is verified as safe, indicating it has not been found in any known data breaches.

Figure 4.26 illustrates the password strength tester menu, where users can test the robustness of their password against four types of attacks: password brute force, AES brute force, dictionary attack, and rainbow table attack. Users can input their own dictionary and rainbow table paths (supporting MD5 .rti2 files from Free Rainbow Tables¹⁷) to customize the attack simulation. Additionally, users can adjust the thread count for multithreaded attack

¹⁷ Free Rainbow Tables. Available at: <https://freerainbowtables.com/>

simulations, providing flexibility for real-time testing. The menu evaluates both the estimated and actual time to crack the password, helping users assess the security of their password against various attack methods.

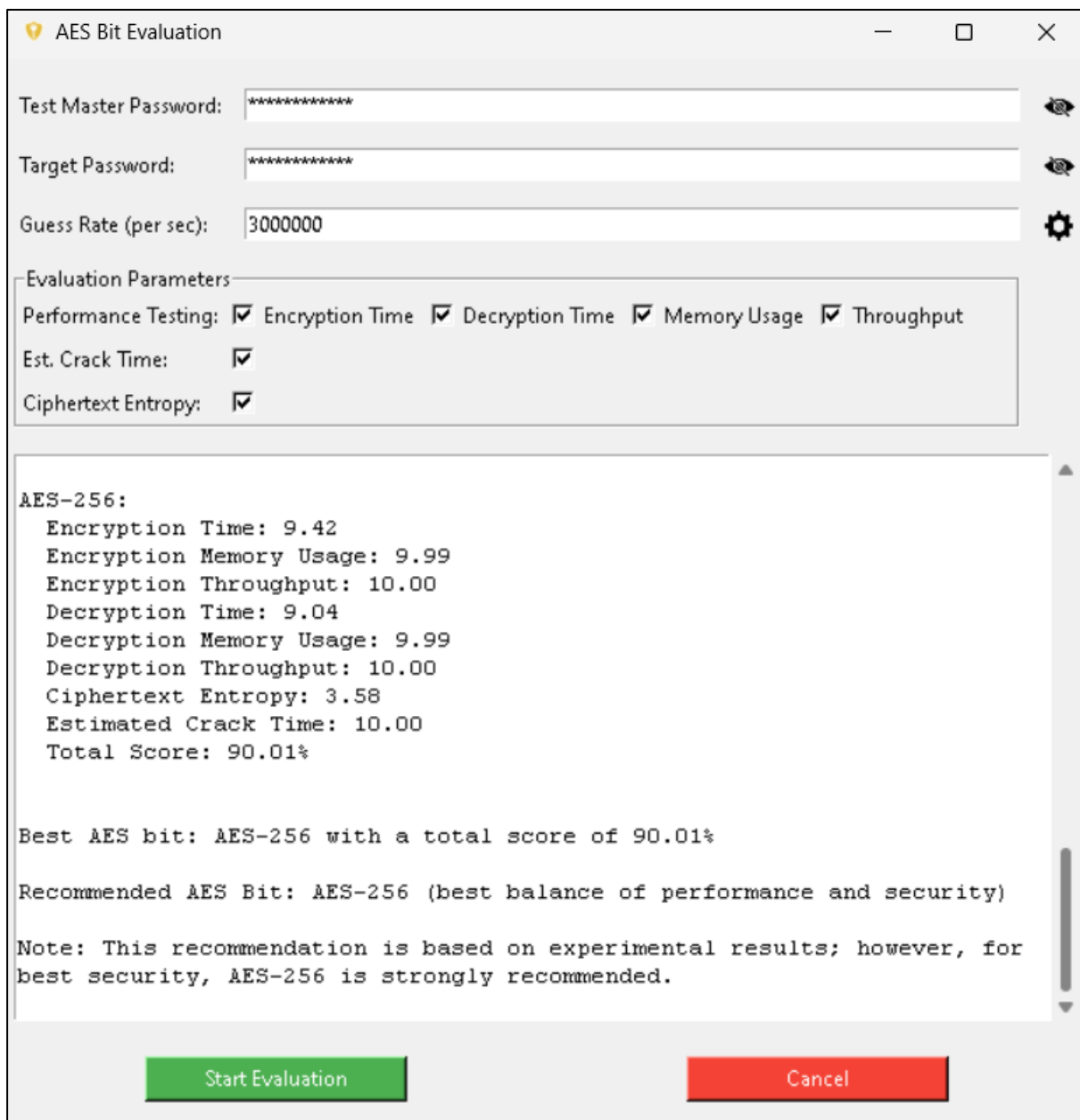
4.3.8 AES Bit Evaluation Menu



Figure 4.27 "Pick For Me" Button

Note. This figure shows the "Pick For Me" button which triggers the AES Bit Evaluation Menu when clicked.

Figure 4.27 displays the "Pick For Me" button located beside the AES bit selection. When clicked, this button triggers the AES bit evaluation process, helping users select the optimal AES bit size for password encryption based on their device's performance and security metrics.



AES Bit Evaluation

Test Master Password:

Target Password:

Guess Rate (per sec):

Evaluation Parameters

Performance Testing: ☒ Encryption Time ☒ Decryption Time ☒ Memory Usage ☒ Throughput

Est. Crack Time: ☒

Ciphertext Entropy: ☒

AES-256:

Encryption Time: 9.42
 Encryption Memory Usage: 9.99
 Encryption Throughput: 10.00
 Decryption Time: 9.04
 Decryption Memory Usage: 9.99
 Decryption Throughput: 10.00
 Ciphertext Entropy: 3.58
 Estimated Crack Time: 10.00
 Total Score: 90.01%

Best AES bit: AES-256 with a total score of 90.01%

Recommended AES Bit: AES-256 (best balance of performance and security)

Note: This recommendation is based on experimental results; however, for best security, AES-256 is strongly recommended.

Start Evaluation **Cancel**

Figure 4.28 AES Bit Evaluation Menu

Note. This figure illustrates the AES bit evaluation menu, where users can configure performance and security metrics for different AES bit sizes to determine the optimal bit for encryption based on their device's performance.

Figure 4.28 displays the AES bit evaluation menu, where users can assess their device's performance and security metrics for different AES bit sizes. Users can input a test master password, test password, configure guess per second settings, and select various performance

metrics such as encryption and decryption time, memory usage, throughput, and security metrics like estimated time to crack and ciphertext entropy. After configuring these parameters, the user can press the "Start" button to run the evaluation, and the system will evaluate the selected AES bit options, displaying the best bit for optimal performance and security based on the results.

4.3.9 Guess Configuration Menu

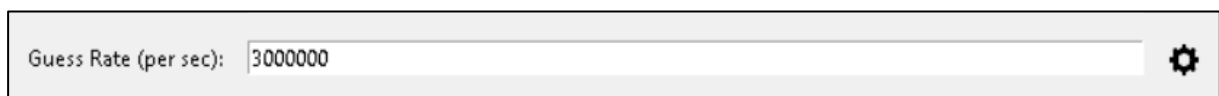


Figure 4.29 Guess Per Second Button

Note. This figure shows the "Configure Guess Per Second" button, which opens the Guess Configuration Menu when clicked.

Figure 4.29 displays the "Configure Guess Per Second" button, represented by a gear icon, located beside the guess per second field. This button can be found in both the password strength tester menu and the AES bit evaluation menu. When clicked, it opens the Guess Configuration Menu, allowing users to configure the number of threads and guess threshold to test their device's password cracking speed.

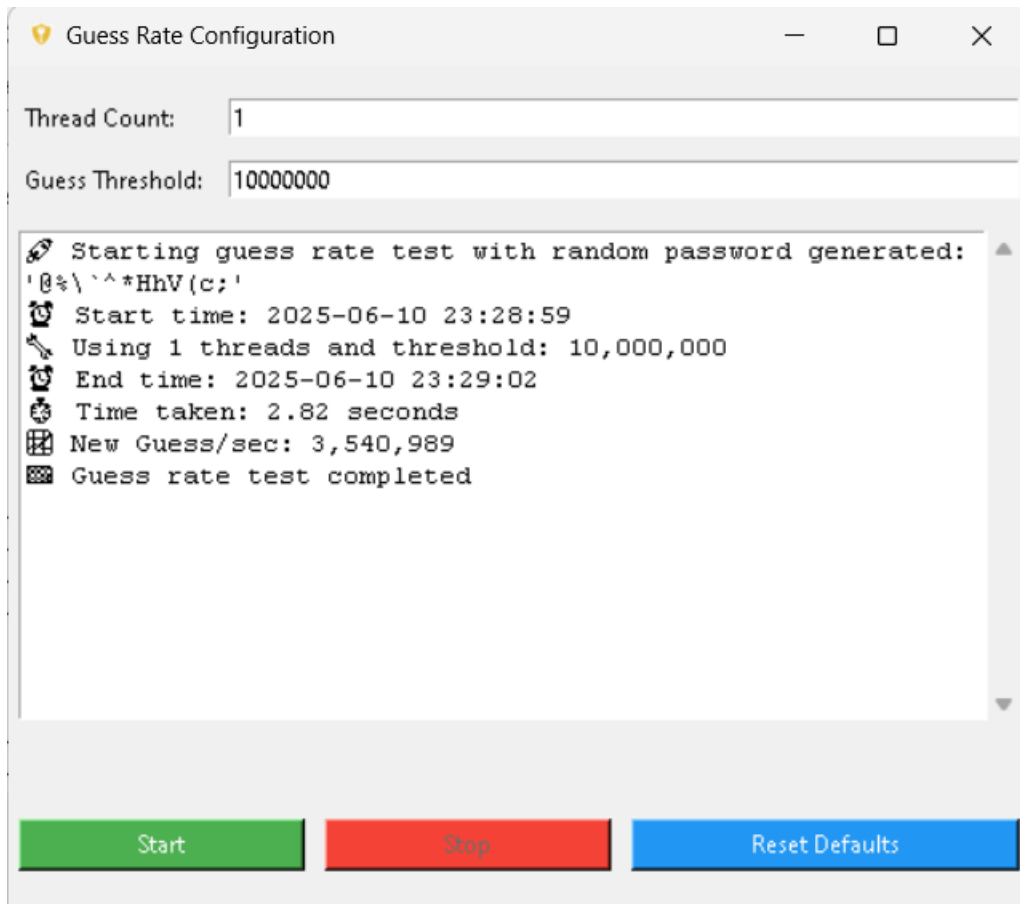


Figure 4.30 Guess Configuration Menu

Note. This figure displays the guess configuration menu, where users can adjust the number of threads and guess threshold to test their device's password cracking speed and calculate the actual guess per second, aiding in both password strength testing and AES bit evaluation.

Figure 4.30 displays the guess configuration menu, where users can configure the number of threads and the guess threshold to test their device's password cracking speed. By adjusting these parameters, the system performs a test to calculate the actual guess per second, which is then used for both password strength testing and AES bit evaluation. This allows users to accurately measure their device's performance and determine their password strength, as well as the optimal AES bit size based on real-world cracking capabilities.

4.3.10 Password Health Checker Menu

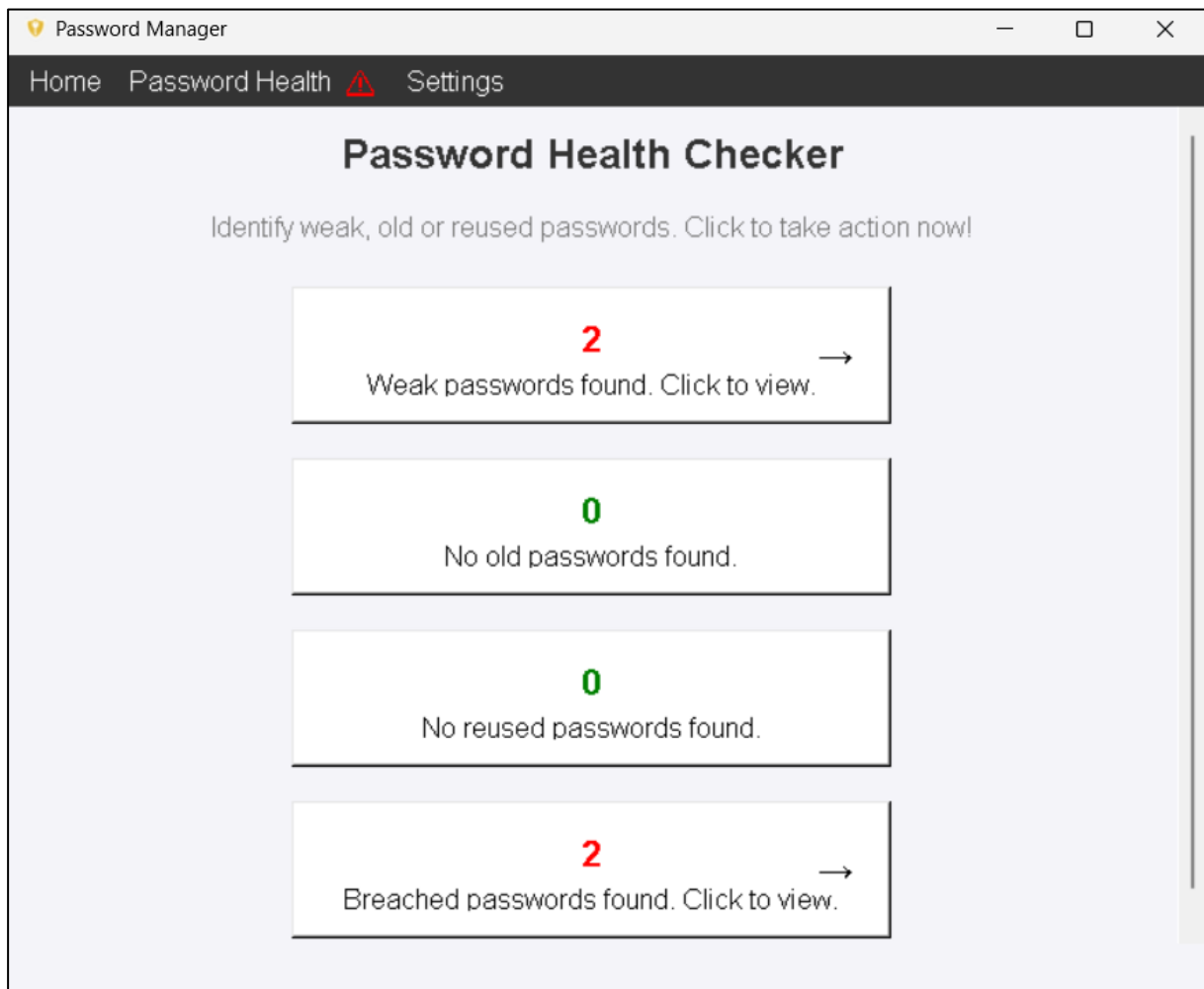


Figure 4.31 Password Health Checker Menu

Note. This figure displays the Password Health Checker menu, where users can view passwords categorized as weak, old, reused, or breached, and click on any category to see more details of affected passwords.

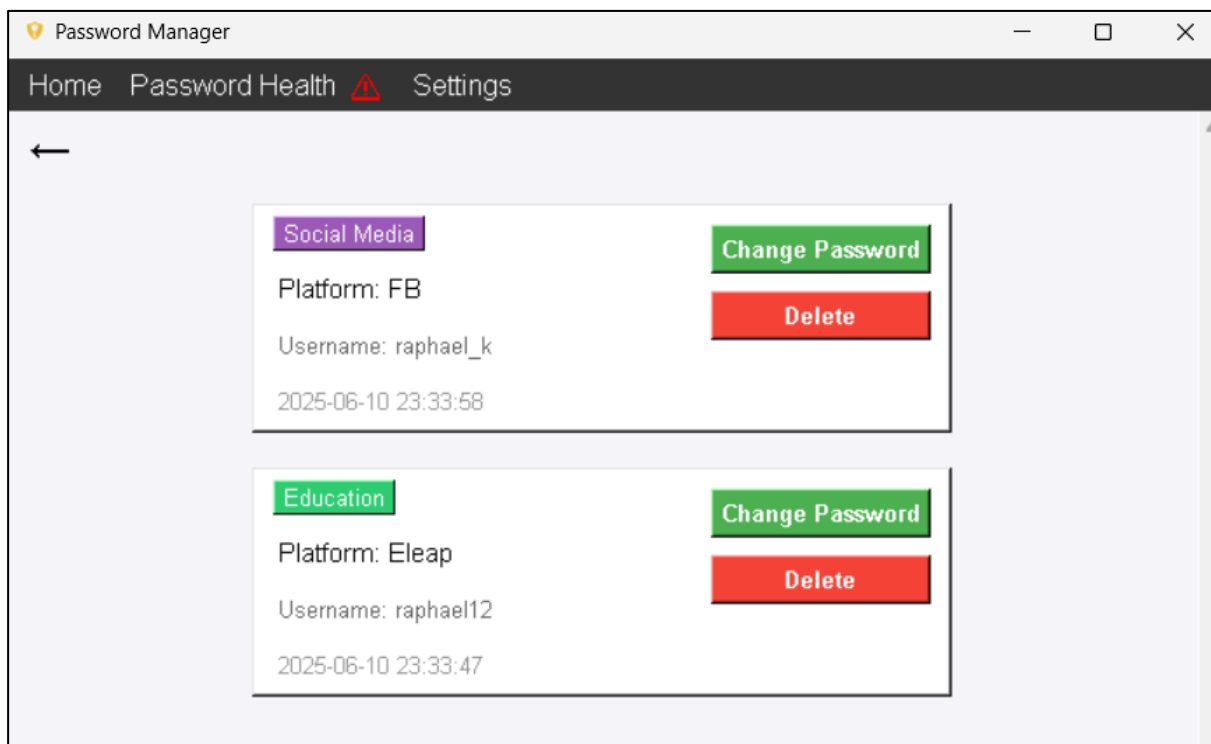


Figure 4.32 Password with Issues Selected

Note. This figure shows the list of passwords with identified issues from the selected category, where users can choose to either fix the issue by clicking "Change" or delete the password.

Figure 4.31 illustrates the Password Health Checker menu, which categorizes passwords into four main categories: weak, old, reused, and breached passwords. Each category shows the number of passwords that fall under it, giving users an overview of the health of their stored passwords. By clicking on any category, users can view a list of passwords with identified issues, as shown in Figure 4.32. From this list, users have the option to press "Change" to update or strengthen the password to resolve the issue, or they can select "Delete" to remove the compromised password entirely. This feature allows users to easily manage and improve the security of their passwords.

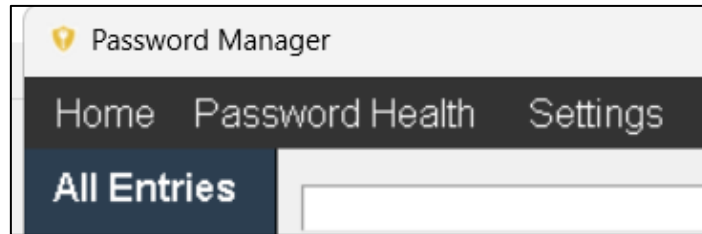


Figure 4.33 Normal Navigation Bar

Note. This figure displays the normal navigation bar, where the "Password Health" menu appears without any issues detected.

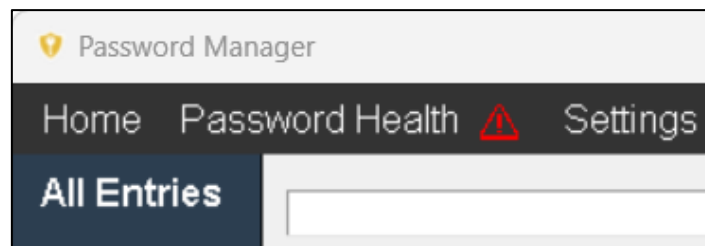


Figure 4.34 Navigation Bar with Password Issues

Note. This figure illustrates the navigation bar with a "Password Health ⚠️" label, indicating that issues have been detected within the user's passwords.

The Password Health Checker feature runs continuously in the background, actively scanning all stored passwords for any potential issues, such as weak, old, reused, or breached passwords. As the system monitors for these issues, it provides real-time updates. If any problems are detected, the navigation bar at the top is updated to display "Password Health ⚠️", signaling that there are issues to address, as shown in Figure 4.34. This serves as a constant reminder for users to review their passwords and ensure they are secure. When no issues are detected, the navigation bar remains in its normal state, as depicted in Figure 4.33, providing a clear visual cue that all passwords are in good health and no immediate actions are required.

4.3.11 Settings Menu

Configure MFA, system alerts, generate recovery keys, backup settings, auto logout time, and clipboard duration.

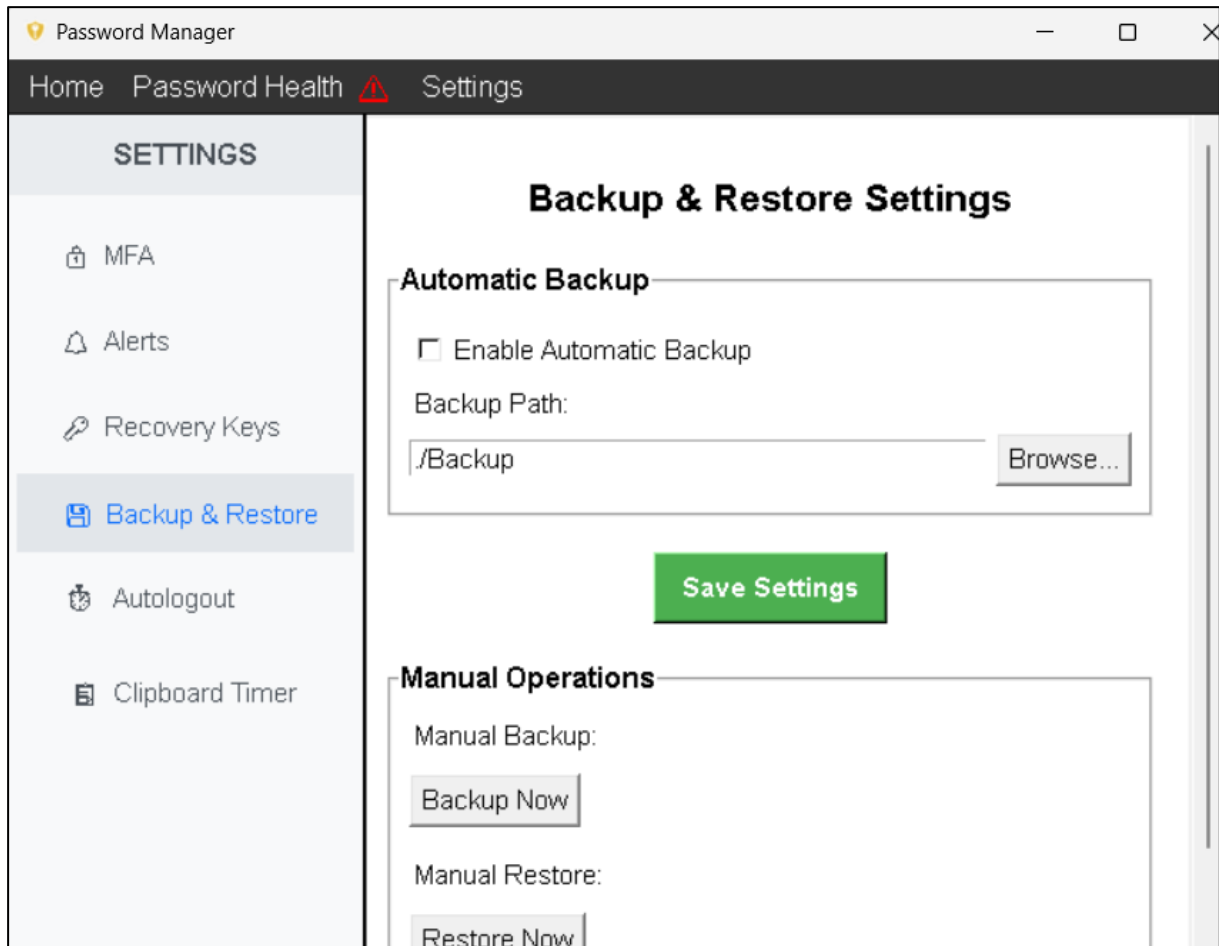


Figure 4.35 Settings Menu

Note. This figure displays the Settings Menu, where users can configure system settings such as MFA, alerts, recovery keys, backup settings, auto logout time, and clipboard duration to personalize security and system functionality.

Figure 4.35 illustrates the Settings Menu, which provides various options for users to configure important system settings. This includes enabling and configuring Multi-Factor Authentication (MFA), setting up system alerts, generating recovery keys, managing backup settings, adjusting the auto logout time, and specifying the clipboard duration. The menu allows users to customize their security preferences and system behavior to enhance both functionality

and protection. Each option can be easily accessed and configured according to the user's needs, ensuring the system aligns with their preferred security and operational standards.

4.4 Summary

In summary, this chapter details the implementation of the password management system, detailing hardware specifications, software setup, and key functionalities. It covers the use of Visual Studio Code, SQLite with pysqlitecipher for encryption, and essential libraries like pyotp for MFA, along with code snippets for password encryption, MFA, password health checks, and AES bit evaluation. The system layout includes an interface with features like password generation, AES bit evaluation, breach detection, and configurable settings menu for enhanced security and usability.

CHAPTER 5: TESTING

5.1 Overview

This chapter presents Phase 4 of the Waterfall methodology: Testing of the Password Management System. It covers the testing processes employed to validate the system's functionality, performance, security, and usability. The tests include functional testing, which consists of unit testing and User Acceptance Testing (UAT), and non-functional testing, which includes performance and security testing. These tests ensure the system meets its design specifications and performs reliably under various conditions. The subsequent sections detail the different types of testing conducted.

5.2 Functional Testing

Functional testing is crucial to ensure that the system behaves as expected and meets the specified requirements. It includes validating individual components through unit testing and assessing the system's performance from the user's perspective through UAT.

5.2.1 Unit Testing

Unit testing is a critical phase in the development process, designed to validate individual components or modules of the system to ensure they function correctly in isolation. Each function and method in the Password Management System was tested independently to confirm their expected behavior. Table 5.1 to 5.8 shows the results of unit testing:

Table 5.1 User Registration Function Test Case

Test Case ID: 1					
Test Objective: To test user registration function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
R001	Register with Master Password	Enter master password: "Testing12()", click on 'OK' button.	User should be able to register into the system	User successfully registers into the system and is redirected to the homepage.	Pass
R002	Register without Password	No input for master password.	User should not be able to register into the system.	User is not allowed to register, and an error message "Master password is required." is shown.	Pass

Note. This table outlines the test cases for validating the user registration function, which involves checking if users can successfully register by entering the master password or if they are appropriately blocked when the master password is not entered. The test ensures that the system properly handles valid and invalid inputs, providing the expected outcomes for both successful and unsuccessful registrations.

Table 5.2 User Login Function Test Case

Test Case ID: 2					
Test Objective: To test user login function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
L001	Login with Valid Master Password	Enter valid master password: "Testing12()", click on 'Login' button.	User should be able to log into the system. If MFA is turned on, user should be redirected to the OTP verification menu. If MFA is off, user should	User logs into the system and is redirected to the OTP verification menu if MFA is turned on, otherwise to the homepage.	Pass

			be redirected to the homepage.		
L002	Login with Invalid Master Password	Enter invalid master password: "Testing34()".	User should not be able to log into the system.	User is not allowed to log into the system, and an error message "Invalid password" is shown.	Pass
L003	Login with Empty Master Password	No input for master password.	User should not be able to log into the system.	User is not allowed to log into the system, and an error message "No master password entered" is shown.	Pass
L004	Login with Valid Recovery Key	Enter valid recovery key: "8UIYNX4U", click on 'Login' button.	User should be able to log into the system. If MFA is turned on, user should be redirected to the OTP verification menu. If MFA is off, user should be redirected to the homepage.	User logs into the system and is redirected to the OTP verification menu if MFA is turned on, otherwise to the homepage.	Pass
L005	Login with Invalid Recovery Key	Enter invalid recovery key: "W35B0709".	User should not be able to log into the system.	User is not allowed to log into the system, and an error message "Invalid recovery key" is shown.	Pass

L006	Login with Empty Recovery Key	No input for recovery key.	User should not be able to log into the system.	User is not allowed to log into the system, and an error message "No recovery key entered" is shown.	Pass
L007	Lockout After Multiple Failures	Enter invalid master password/recovery key after 5 failed attempts.	User should be locked out and presented with a cooldown period of 1 hour before trying again.	User is locked out and presented with a 1-hour cooldown period, and a message "Too many failed attempts" is shown.	Pass

Note. This table presents test cases designed to validate the user login function, ensuring that users can log in successfully with valid master password and are blocked from logging in with invalid master password. It also includes tests for handling recovery key, and the lockout mechanism after multiple failed login attempts. The system's response to different input scenarios is verified to ensure correct functionality.

Table 5.3 OTP Verification Test Case

Test Case ID: 3					
Test Objective: To test the OTP verification function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
O001	Verify with Valid OTP	Enter valid OTP: "661990" from authenticator app.	User should be successfully authenticated and redirected to the homepage.	User is authenticated and redirected to the homepage.	Pass
O002	Verify with Invalid OTP	Enter invalid OTP: "123456".	User should not be authenticated, and an error	User is not authenticated, and an error message	Pass

			message “Invalid OTP” should be shown.	“Invalid OTP” is displayed.	
O003	Verify with Empty OTP	No input for OTP.	User should not be authenticated, and an error message “OTP is required” should be shown.	User is not authenticated, and an error message “OTP is required” is displayed.	Pass
O004	Lockout After Multiple Failed OTPs	Enter invalid OTP after 5 failed attempts.	ser should be locked out and presented with a 1-hour cooldown period before trying again.	User is locked out and presented with a 1-hour cooldown period, and a message "Too many failed attempts" is shown.	Pass

Note. This table presents test cases for validating the OTP (One-Time Password) verification function, ensuring that users can successfully authenticate via OTP, receive appropriate error messages for invalid input or no input, and are locked out after 5 failed attempts for added security. The system's handling of OTP validation is tested to confirm its reliability and user authentication flow.

Table 5.4 Password Encryption and Decryption Test Case

Test Case ID: 4					
Test Objective: To test password encryption and decryption functions.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
E001	Encrypt Password with AES	password = "test123"	Password is encrypted using AES with the selected bit size (128/192/256) and key derived from the master password	Password encrypted correctly.	Pass

			using PBKDF2.		
E002	Decrypt Password with Toggle	No input data. Click on show/hide button on the password field to view the decrypted password.	Password is decrypted correctly.	Password decrypted correctly.	Pass

Note. This table outlines the test cases for validating the password encryption and decryption functions, ensuring that passwords are securely encrypted using AES with the selected bit size and decrypted correctly using the show/hide functionality. The encryption key is derived from the master password using PBKDF2, and the system's ability to securely store and reveal passwords is tested.

Table 5.5 Password Generation Test Case

Test Case ID: 5					
Test Objective: To test the password generation function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
G001	Generate Password with Default Criteria	Click on "Generate Password" button with default criteria (length = 12, includes digits, upper/lowercase letters and special characters).	Password is generated with the specified criteria (length = 12, includes digits, upper/lowercase letters and special characters).	Password generated correctly.	Pass
G002	Generate with Custom Criteria	Click on "Generate Password" button with customized criteria (length = 16, includes digits, lowercase letters, and special characters).	Password is generated with the customized criteria.	Password generated correctly.	Pass

G003	Generate with Invalid Length	Click on "Generate Password" button with invalid criteria (length = -1).	Error message "Invalid length" is displayed.	Error message displayed.	Pass
G004	Generate with No Criteria	Click on "Generate Password" button without selecting any criteria.	Default criteria (length = 12, includes digits, upper/lowercase letters, special characters) is used to generate the password.	Password generated with default criteria.	Pass

Note. This table presents the test cases for validating the password generation functionality, ensuring that the system correctly generates passwords based on both default and customized criteria, and handles invalid or missing criteria appropriately.

Table 5.6 Password Breach Check Test Case

Test Case ID: 6					
Test Objective: To test the password breach check function.					
Test ID	Test Case Name	Input Datar	Expected Result	Actual Result	Status (Pass/Fail)
B001	Check Breached Password	Enter a password that has been previously breached (e.g., "123456").	The system checks the password against the Pwned Passwords API and displays a warning message indicating the password has been exposed in data breaches.	Warning message displayed: "This password has been seen in data breaches. You should change it."	Pass
B002	Check Safe Password	Enter a password that has not been breached (e.g., "UniquePassword123!#").	The system checks the password and confirms that it has not been exposed in any	Message displayed: "Good news — this password has not	Pass

			known data breaches.	been found in any known data breaches."	
B003	Check with Empty Input	No input for the password and click on the breach check button (represented by tick icon).	The system shows an error message saying "No password entered" and does not proceed with the check.	Error message displayed: "No password entered."	Pass

Note. This table outlines the test cases for validating the password breach check function, ensuring that the system correctly identifies breached passwords using the "Have I Been Pwned" API, handles unentered passwords appropriately, and displays accurate warnings or success messages based on the breach status.

Table 5.7 Password Strength Testing Test Case

Test Case ID: 7					
Test Objective: To test the password strength testing function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
S001	Password Brute Force Attack Simulation	Enter a password: "Ab2!", select "Password Brute Force" attack and click "Start" to perform the actual attack.	The system displays the estimated time to crack based on password brute force attack and allows the actual attack to be performed.	Estimated time to crack shown, and actual password brute force attack is performed.	Pass
S002	AES Brute Force Attack	Enter a password: "Ab2!", select "AES Brute Force" attack and click "Start" to perform the actual attack.	The system displays the estimated time to crack based on AES brute force and performs the attack. A warning is displayed indicating AES brute force is	Estimated time to crack shown, warning displayed about infeasibility, and AES brute force attack performed.	Pass

			computationally infeasible with current technology.		
S003	Dictionary Attack	Enter a password: "Ab2!", select "Dictionary Attack," provide the dictionary path, and click "Start."	The system checks the password against the dictionary, estimates the time to crack, and performs the actual dictionary attack. If the password is found, it shows the time taken.	Estimated time to crack displayed, and actual attack is performed with dictionary path provided.	Pass
S004	Rainbow Table Attack	Enter a password: "Ab2!", select "Rainbow Table Attack," provide the rainbow table	The system estimates the time to crack the password using the rainbow table attack method and performs the attack using the rainbow table if available.	Estimated time to crack displayed, and the actual rainbow table attack is performed using the specified rainbow table path.	Pass
S005	Password Strength Test with Empty Password	No input for the password and click on the "Start" button.	The system shows an error message indicating that no password was entered.	Error message "No password entered" is displayed.	Pass

Note. This table outlines the test cases for validating the password strength testing function, ensuring that the system can estimate and perform attacks (password brute force, AES brute force, dictionary attack, and rainbow table attack) on the password to evaluate its security. The tests verify that the estimated time to crack is displayed and that the corresponding attack is executed.

Table 5.8 AES Bit Evaluation Test Case

Test Case ID: 8 Test Objective: To test the AES bit evaluation function.					
Test ID	Test Case Name	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
A001	Evaluate AES Bit Performance	Enter test master password: "Testing12()" and test password: "Ab2!", use default guess rate: 3000000, select all the evaluation parameters and click "Start" to evaluate AES bit options.	The system runs the AES bit evaluation, testing encryption and decryption performance (time, memory, throughput) and security metrics (estimated crack time, ciphertext entropy). The optimal AES bit size (128, 192, or 256) is displayed based on the results.	AES evaluation completed, best bit selected based on performance and security results.	Pass
A002	Evaluate AES Without Password	No test password entered and click "Start" to evaluate AES bit options.	The system should display an error message, such as "No password entered," and not perform the AES bit evaluation.	Error message "No password entered" displayed, and AES bit evaluation not performed.	Pass

Note. This table outlines the test cases for evaluating the AES bit options (128, 192, and 256) based on performance and security metrics. The tests ensure that the system correctly performs the evaluation by testing encryption and decryption performance as well as security aspects like estimated crack time and ciphertext entropy. Additionally, the table checks if proper error messages are displayed when no password is entered for evaluation.

5.2.2 User Acceptance Testing

User Acceptance Testing (UAT) was conducted to ensure that the Password Management System meets the needs and expectations of the end users. This phase involved evaluating the usability and functionality of the system through user interactions. A Google Form survey, as detailed in Appendix I, was conducted with 23 respondents to gather feedback regarding their experience with the system, helping to assess the system's effectiveness, ease of use, and overall satisfaction. The results of the UAT are presented in Figures 5.1 to 5.10, highlighting key feedback and user insights.

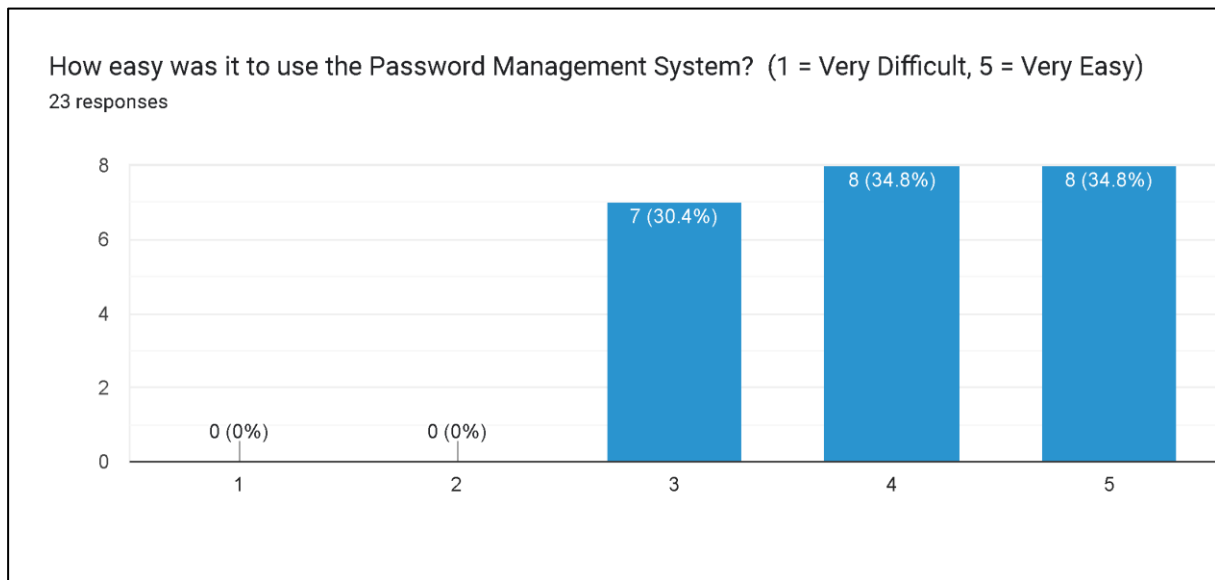


Figure 5.1 Ease of Use

Note. This figure illustrates the feedback on the system's ease of use, with 34.8% of users rating it as "Very Easy," 34.8% rating it as "Easy," and 30.4% selecting "Neutral".

As shown in Figure 5.1, the majority of users found the system easy to use, with 69.6% selecting "Very Easy" or "Easy." However, 30.4% of users were neutral about the ease of use, indicating that while most users were able to navigate the system without major issues, there is still some variation in the user experience. This neutral feedback suggests that certain aspects of the system's interface or functionality could be further optimized to enhance usability for a

broader range of users. Despite the overall positive response, this highlights an opportunity for refinement in specific areas to ensure a more consistent experience for all users.

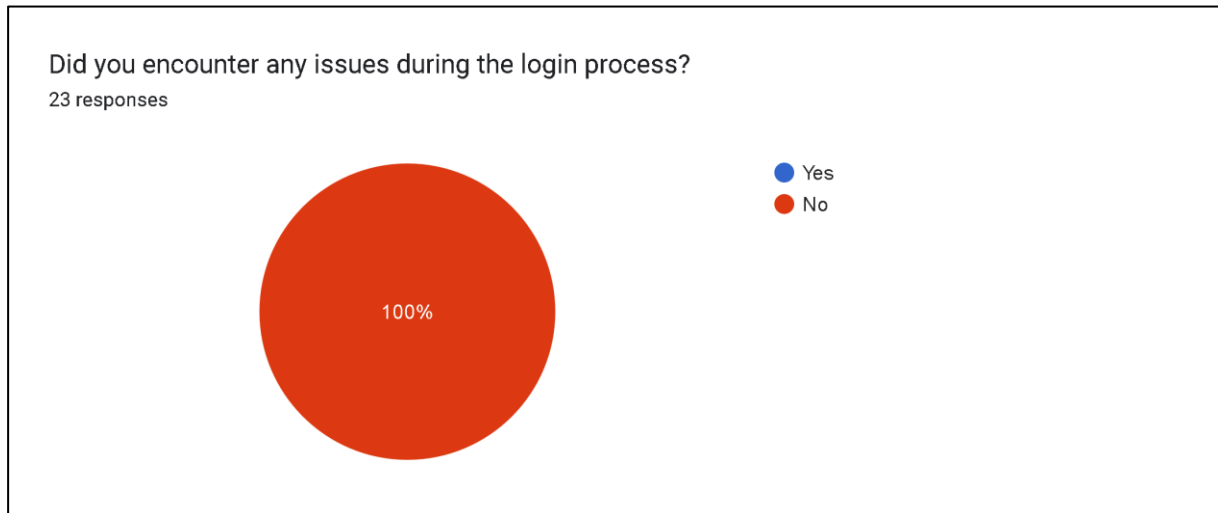


Figure 5.2 Login Process

Note. This figure shows the results of the login process testing, where 100% of users successfully logged in without encountering any issues.

From Figure 5.2, all users successfully logged into the system without issues, indicating that the login process was seamless and functioning as expected. The absence of any reported problems suggests that the authentication process is stable and reliable. Users were able to access the system without encountering any barriers, which confirms the system's robust performance in handling user credentials.

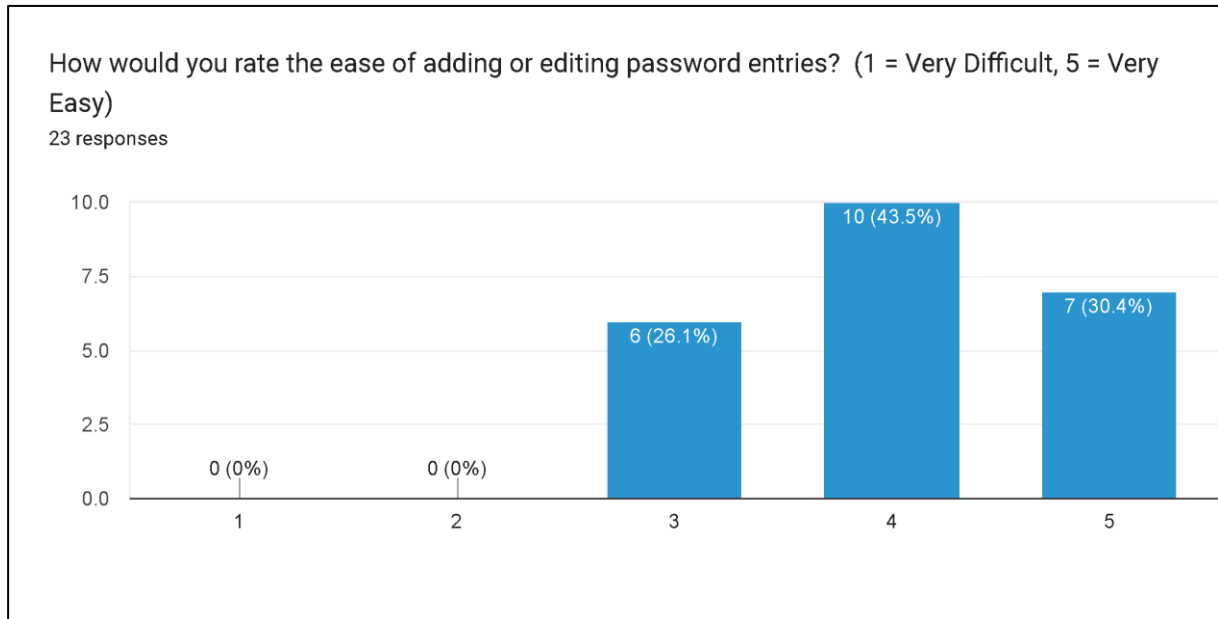


Figure 5.3 Ease of Adding/Editing Password Entries

Note. This figure presents feedback on adding and editing password entries, where 73.9% found it either "Easy" or "Very Easy," with 15% suggesting improvements, particularly around usability.

According to the results in Figure 5.3, 26.1% of users rated the process of adding and editing password entries as "Neutral," 43.5% found it "Easy," and 30.4% rated it as "Very Easy." While most users had a positive experience, 15% of users suggested improvements, with some finding the process slightly cumbersome. Feedback from users pointed out that the interface could be more intuitive, and additional options or a more streamlined approach might enhance the user experience.

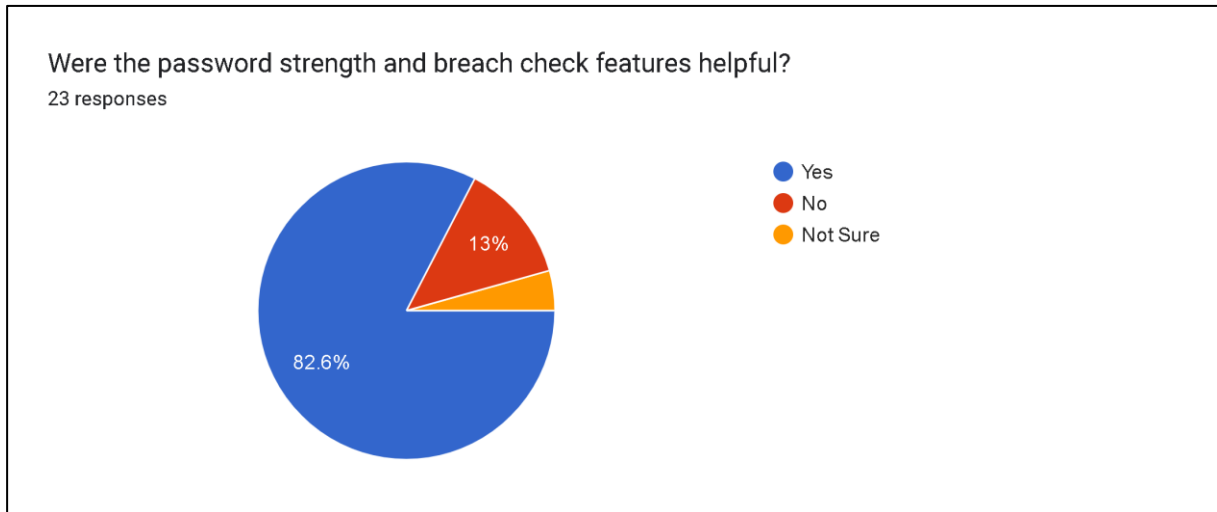


Figure 5.4 Password Strength and Breach Check

Note. This figure highlights the user feedback on the password strength and breach check features, where 82.6% found them helpful in assessing password security.

Figure 5.4 indicates that 82.6% of users found the password strength and breach check features helpful in assessing their password security. These features provided valuable insights, allowing users to assess whether their passwords were strong enough and whether they had been exposed in data breaches. 13% of users did not find these features useful, possibly due to a lack of understanding of how to interpret the results or insufficient instructions. Additionally, 4.3% were unsure, expressing the need for clearer guidance on how to effectively utilize these features, which suggests room for improvement in the user interface or educational resources to make the security insights more accessible and actionable for users.

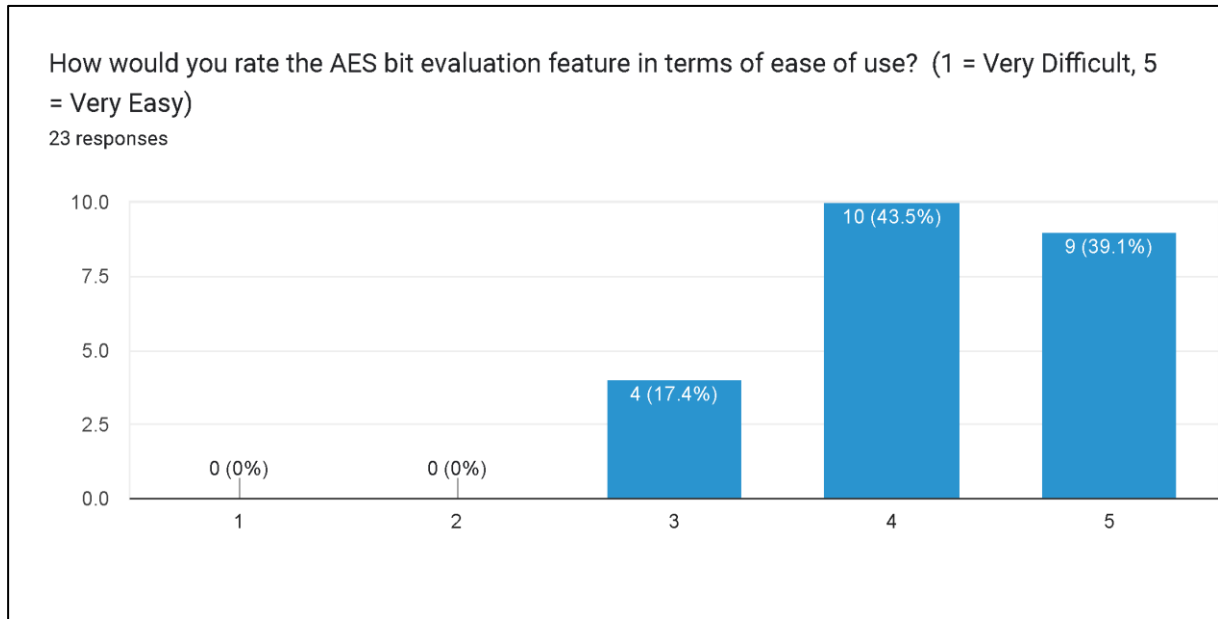


Figure 5.5 AES Bit Evaluation

Note. This figure displays the results of the AES bit evaluation feature, with 17.4% of users rating it as "Neutral," 43.5% finding it "Easy," and 39.1% rating it as "Very Easy."

Figure 5.5 displays the results of the AES bit evaluation feature, where 17.4% of users rated it as "Neutral," 43.5% found it "Easy," and 39.1% rated it as "Very Easy." While the majority of users were able to use the feature without issues, some found it could be simplified for an even better user experience. A small portion of users suggested improvements to make the feature more accessible, particularly for less technical users. This feedback indicates that while the feature is generally well-received, further refinement could enhance usability for a broader audience.

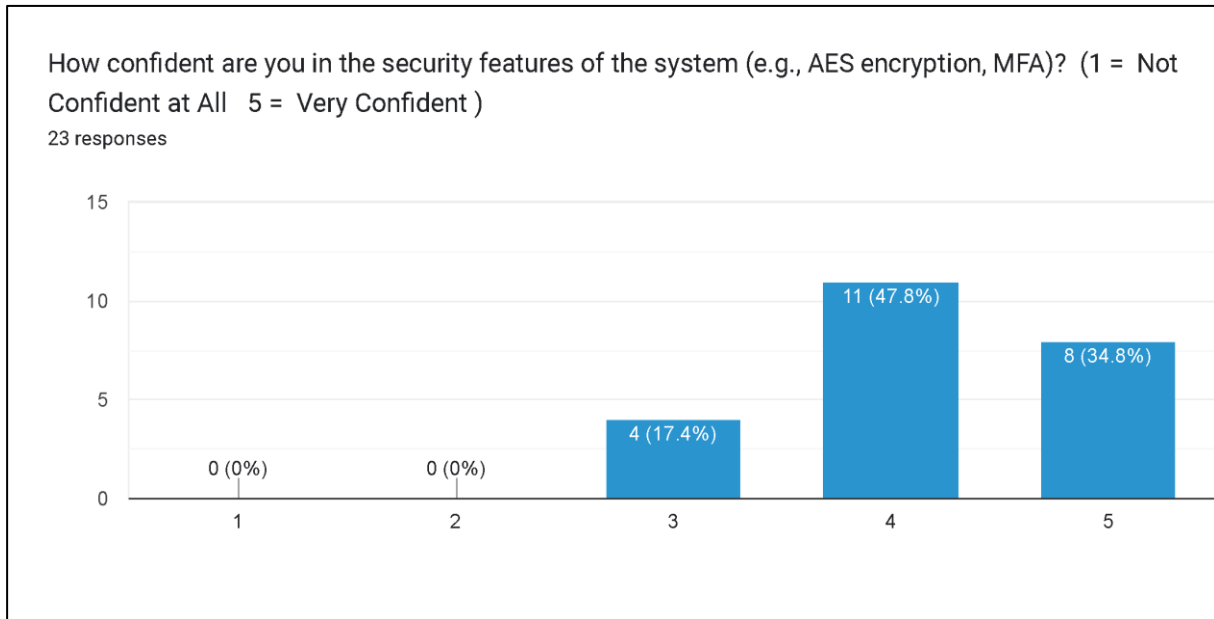


Figure 5.6 Confidence in Security Features

Note. This figure shows users' confidence in the system's security features, with 17.4% of users feeling "Neutral," 47.8% expressing "Confident," and 34.8% being "Very Confident" in the security measures, including AES encryption and MFA.

From Figure 5.6, the majority of users (82.6%) expressed confidence in the system's security features, with 47.8% feeling "Confident" and 34.8% being "Very Confident" in the AES encryption and MFA. However, 17.4% of users were "Neutral," indicating that while most users trust the system's security, there is a segment of users who may require more information or reassurance to feel fully confident in the system's protective measures. This suggests that clearer communication or additional guidance on security features could help increase user trust and confidence further.

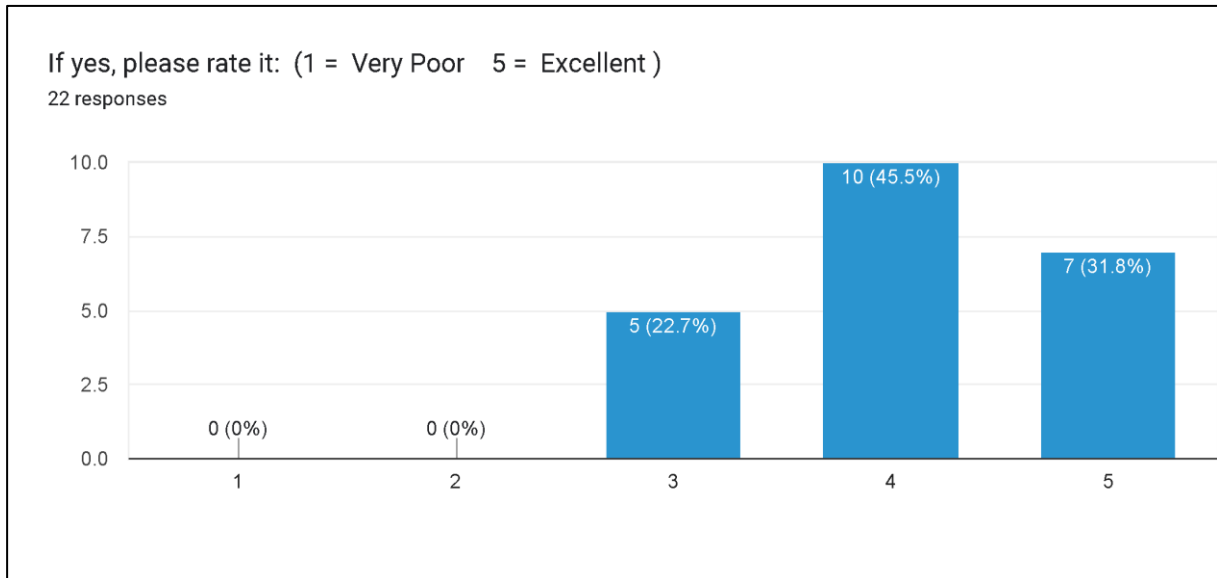


Figure 5.7 Generate Password Feature

Note. This figure presents feedback on the "Generate Password" feature, with 77.3% of users rating it as "Good" or "Excellent" for its convenience, while 22.7% remained neutral.

From Figure 5.7, 77.3% of users rated the "Generate Password" feature as "Good" or "Excellent," with 45.5% rating it as "Good" and 31.8% rating it as "Excellent." However, 22.7% of users were neutral, indicating that while the feature is generally appreciated for its convenience, some users may not find it as useful or intuitive. This suggests that there is room for further improvement in the user interface or functionality of the feature, such as providing more customization options or clearer instructions to enhance its overall value for all users.

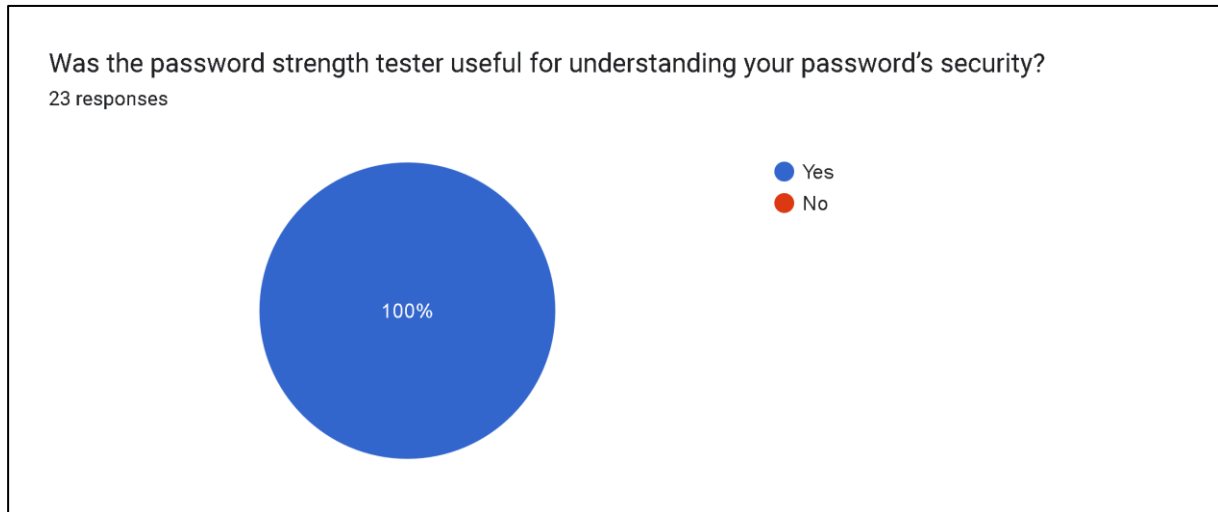


Figure 5.8 Password Strength Tester

Note. This figure illustrates user feedback on the password strength tester, with 100% of users finding it useful for evaluating password security.

As shown in Figure 5.8, all users found the password strength tester useful, with many commenting that it helped them better understand their password security. Some users suggested that clearer instructions could enhance the effectiveness of the feature, indicating that while the tool was useful, additional guidance would further improve its usability.

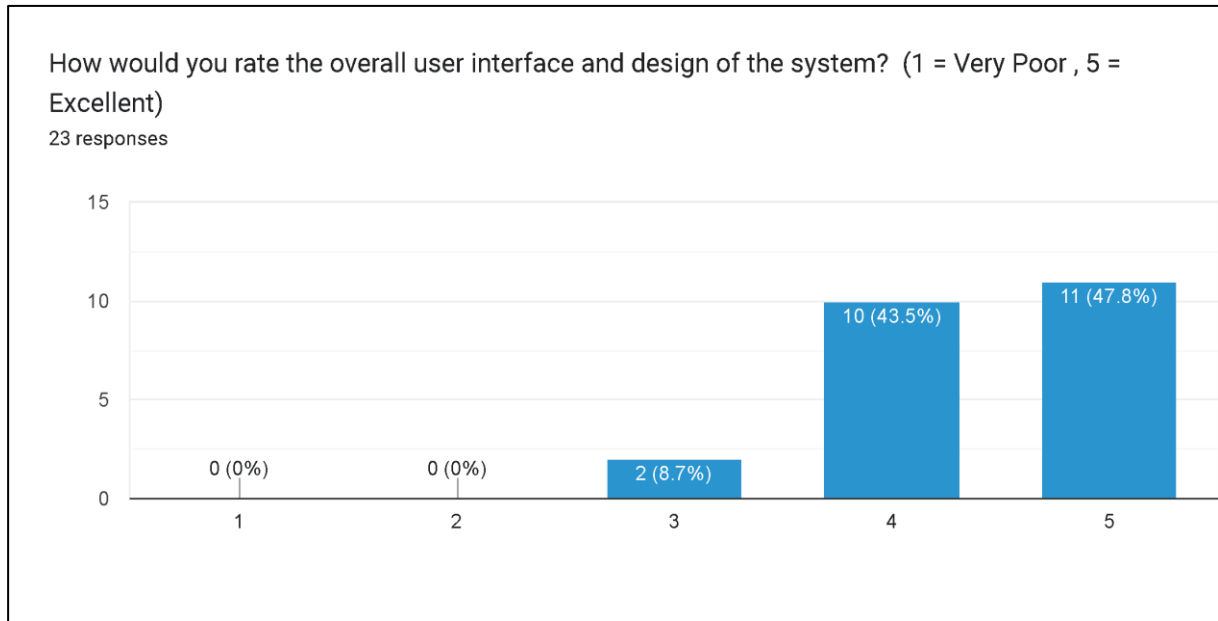


Figure 5.9 User Interface and Design

Note. This figure presents users' ratings of the system's user interface and design, with 47.8% rating it as "Excellent," 43.5% as "Good," and 8.7% as "Neutral".

As shown in Figure 5.9, 8.7% of users rated the system's user interface and design as "Neutral," 43.5% rated it as "Good," and 47.8% rated it as "Excellent." While the majority of users found the design to be positive, the feedback suggests that further refinement could enhance the overall user experience.

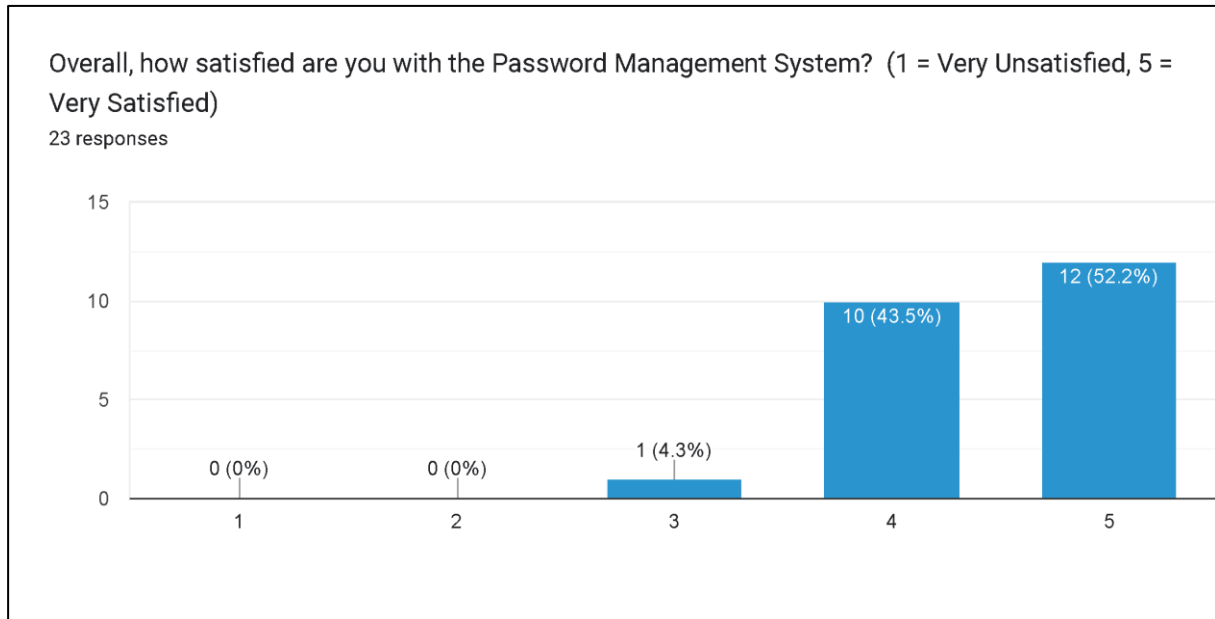


Figure 5.10 Overall Satisfaction

Note. This figure displays the overall satisfaction level of users with the system, with 52.2% being "Very Satisfied," 43.5% "Satisfied," and 4.3% "Neutral,"

From Figure 5.10, 52.2% of users expressed being "Very Satisfied," 43.5% were "Satisfied," and 4.3% were "Neutral," highlighting a strong level of overall satisfaction with the system. However, a small portion of users indicated neutrality, suggesting that while the system met most users' expectations, there may still be room for minor improvements, particularly related to usability.

To summarise the results of the User Acceptance Testing (UAT), it revealed that the Password Management System generally met users' expectations in terms of ease of use, functionality, and security. Most users found the system easy to navigate, with 69.6% rating its usability as "Very Easy" or "Easy." The login process was seamless, with all users successfully logging in without issues, while 73.9% found adding or editing password entries straightforward. The password strength and breach check features were deemed helpful by 82.6% of users, and 75% rated the AES bit evaluation feature as either "Easy" or "Very Easy."

Confidence in the system's security, particularly AES encryption and Multi-Factor Authentication (MFA), was high, with 82.6% of users expressing confidence. The "Generate Password" feature was also highly rated, with 77.3% of users finding it convenient, though some users requested more customization options. The password strength tester received 100% positive feedback, further reinforcing the system's effectiveness in helping users assess their password security. In terms of design, 91.3% of users rated the user interface as "Good" or "Excellent," but some indicated a need for further refinement. Overall, the system received strong approval, with 95.7% of users either "Very Satisfied" or "Satisfied." Nonetheless, there is room for minor improvements, particularly in usability and user guidance.

5.3 Non-Functional Testing

Non-functional testing focuses on evaluating the system's performance, security, and other non-functional aspects, such as usability and reliability. Unlike functional testing, which verifies that the system performs its intended tasks correctly, non-functional testing ensures that the system performs optimally under various conditions and maintains security and stability. In the context of the Password Management System, this includes performance testing to assess how the system handles operations under different loads, and security testing to ensure that the system provides the necessary protections against vulnerabilities and attacks.

5.3.1 Performance Testing

Performance testing was conducted to evaluate the system's behavior under different load conditions. The primary focus was on measuring how quickly the system can handle operations such as encrypting and decrypting passwords, generating new passwords, and simulating password attacks. To comprehensively evaluate the performance, the system's built-

in AES bit evaluation tool was used to assess encryption and decryption metrics after the performance testing test cases. This included testing the encryption and decryption performance for password lengths ranging from 1 to 5 characters, using a constant test master password. The detailed results of this evaluation are presented in Table 5.9 below.

5.3.1.1 Test Cases for Performance Testing

Table 5.9 Performance Testing for Password Management System

Test Case ID: 9 Test Objective: To test performance of the system.						
Test Case ID	Test Case Name	Test Objective	Input Data	Expected Output	Actual Output	Test Result
P001	Stress Testing	To test system's response under high resource usage	Add and edit password entries multiple times with varying password lengths (up to 100 characters) and complexities (including digits, uppercase, lowercase, and special characters).	The system should remain stable and responsive during the addition or editing of passwords with long and complex inputs.	The system remains stable during multiple additions/edits of password entries with long and complex passwords.	Pass
P002	Endurance Testing	To test system's performance under extended usage	During password strength testing, including estimation of time to crack and actual attacks, using long and complex	No memory leaks or performance degradation should occur during the password strength testing with long and	The system operates without memory leaks or performance issues during password strength testing.	Pass

			passwords (up to 100 characters) and performing simulated attacks like brute force and rainbow table attacks.	complex passwords.		
P003	Response Time Testing	To measure system's response time for password-related tasks	Measure the time taken to add, edit, generate passwords (up to 100 characters), and perform strength tests on long and complex passwords.	The system should respond within an acceptable time frame for each operation involving long and complex passwords.	The system meets the expected response time for adding/editing passwords, generating long passwords, and performing tests.	Pass
P004	Scalability Testing	To test the system's stability under increasing complexity	Gradually increase the number of password entries, including long and complex passwords (up to 100 characters), and increase the complexity of password generation and strength testing.	The system should handle the increased load of password operations (with long passwords) without performance degradation.	The system handles more complex password entries and operations involving long passwords without performance issues.	Pass

Note. This table outlines the results of performance testing on the Password Management System, which includes tests for stress, endurance, response time, and scalability. The system was evaluated under different load conditions, including adding and editing long, complex passwords, generating passwords, and performing strength tests. The system demonstrated stability, reliability, and good response times throughout the tests, with no performance degradation or memory leaks observed.

5.3.1.2 Encryption and Decryption Performance Metrics

Following the performance tests, the AES bit evaluation tool was used to assess encryption and decryption metrics for each password length from 1 to 5 characters, across three AES bit sizes: 128, 192, and 256. Figures 5.11 to 5.16 present the results of encryption time, decryption time, memory usage, and throughput for each password length and AES bit configuration.

Input Used in AES Bit Evaluation:

- **Test Master Password:** _086zXg]@/=w (12 characters, consisting of uppercase, lowercase, digits, and special characters)
- **Test Password:** }M2-83j*=0Of (12 characters, consisting of uppercase, lowercase, digits, and special characters)
- **Guess Per Second:** 3,000,000
- **Password Length 1:** Single character (e.g., "a")
Rationale: This represents the simplest possible password, used for the lower bound of password strength testing.
- **Password Length 2:** Two characters (e.g., "a1")
Rationale: A small increase in length and complexity, combining lowercase letters and digits.
- **Password Length 3:** Three characters (e.g., "a1B")
Rationale: Adds uppercase letters to increase complexity, making the password more secure.

- **Password Length 4:** Four characters (e.g., "a1B*")

Rationale: Introduces a special character to the password, further increasing its strength.

- **Password Length 5:** Five characters (e.g., "a1B*9")

Rationale: The length and complexity increase further, using lowercase, uppercase, digits, and special characters to simulate stronger passwords.

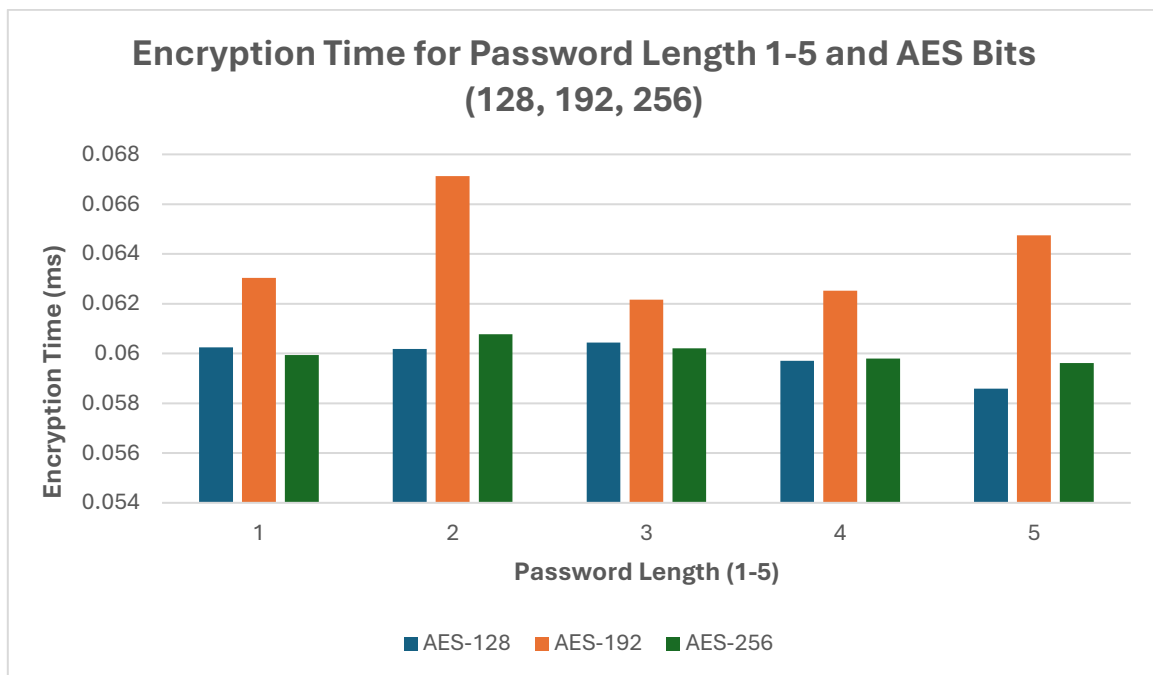


Figure 5.11 Encryption Time for AES Bit Sizes 128, 192, and 256

Note. This figure shows the encryption time for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Encryption time refers to the amount of time required to encrypt a given set of data using a specific encryption algorithm. It is an important metric in assessing the efficiency of encryption methods, with lower encryption times generally being preferred for faster processing. In Figure 5.11, the encryption time for AES bit sizes 128, 192, and 256 across password lengths from 1 to 5 demonstrates that AES-256 performs the fastest for password lengths 1 and 3, while AES-128 outperforms the others for password lengths 2, 4 and 5. Overall, AES-128 emerges as

the most efficient in terms of encryption time. This makes AES-128 particularly suitable for resource-constrained environments where faster encryption speeds are critical, without significantly sacrificing security.

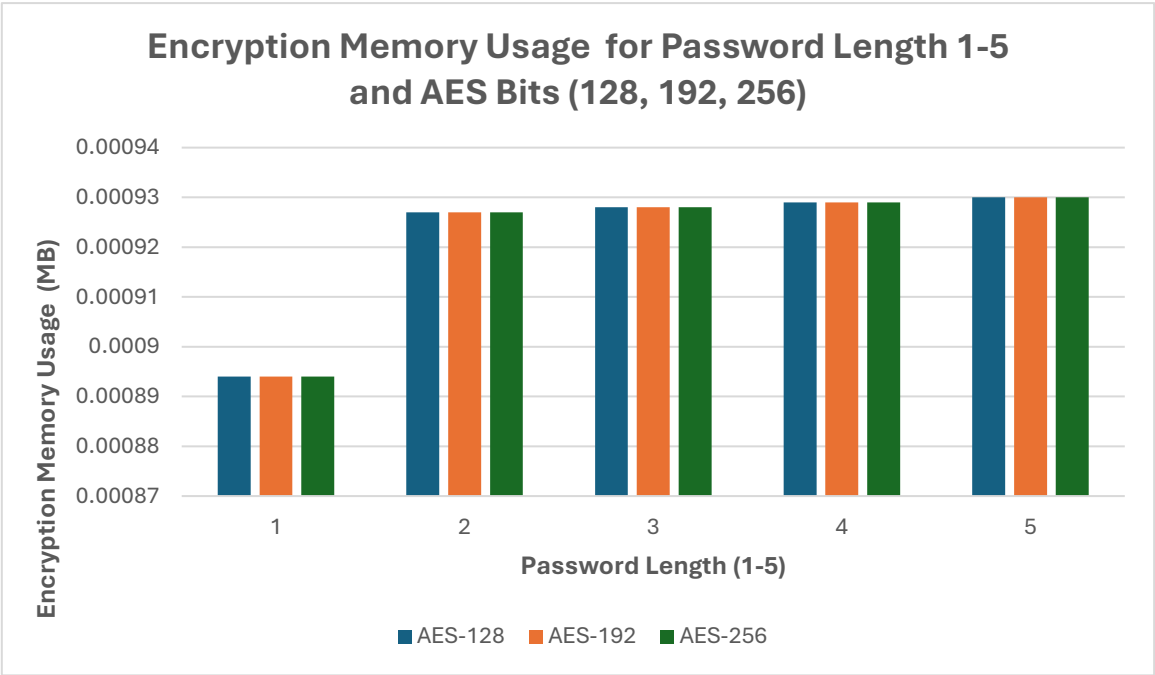


Figure 5.12 Encryption Memory Usage for AES Bit Sizes 128, 192, and 256

Note. This figure shows the encryption memory usage for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Encryption memory usage refers to the amount of memory required by the system to perform the encryption process. This includes the memory needed to store the encryption keys, buffers, and any intermediate data used during encryption. Efficient encryption methods aim to minimize memory consumption while maintaining strong security. In Figure 5.12, the encryption memory usage remains identical across all three AES bit sizes for each password length. This suggests that the choice of AES bit size does not significantly affect memory consumption, allowing users to focus on other factors, such as encryption speed and security, without concerns about increased memory usage.

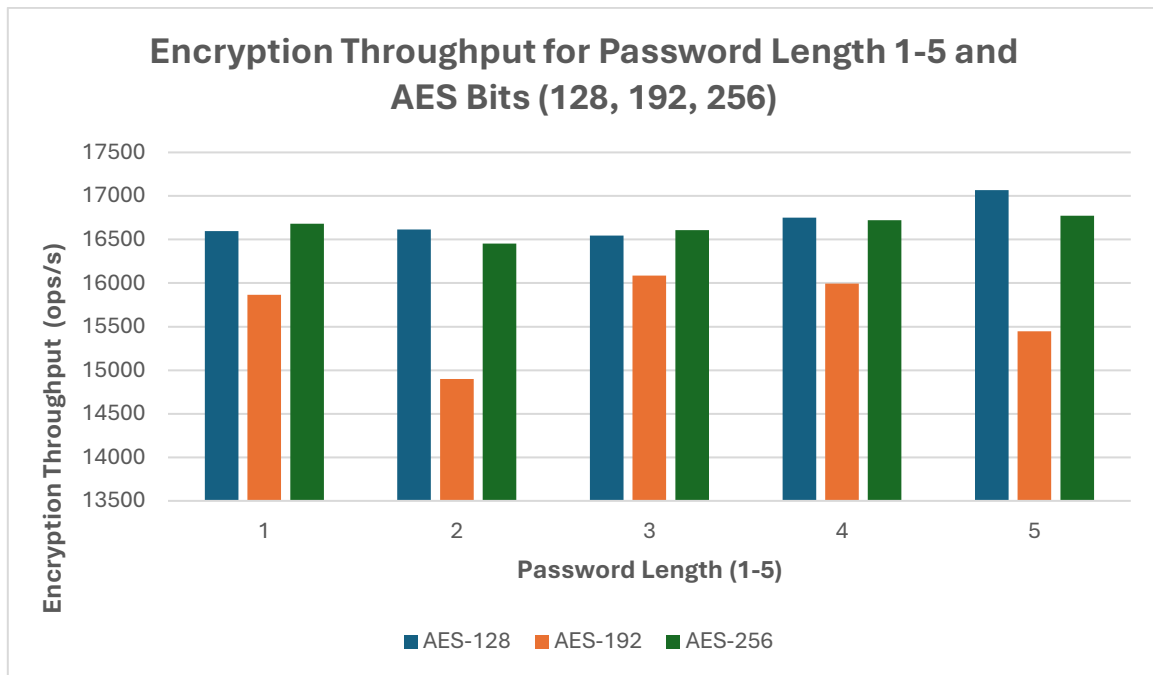


Figure 5.13 Encryption Throughput for AES Bit Sizes 128, 192, and 256

Note. This figure shows the encryption throughput for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Encryption throughput, measured in operations per second (ops/s), reflects the number of encryption operations the system can perform in a given time. As shown in Figure 5.13, AES-256 achieves the highest throughput for password lengths 1 and 3, indicating it is more efficient for shorter passwords. However, AES-128 outperforms the others for password lengths 2, 4, and 5, suggesting that as password length increases, AES-128 provides superior throughput. This makes AES-128 the more suitable choice for scenarios where processing efficiency is critical, particularly when dealing with larger datasets or operating on devices with limited computational resources.

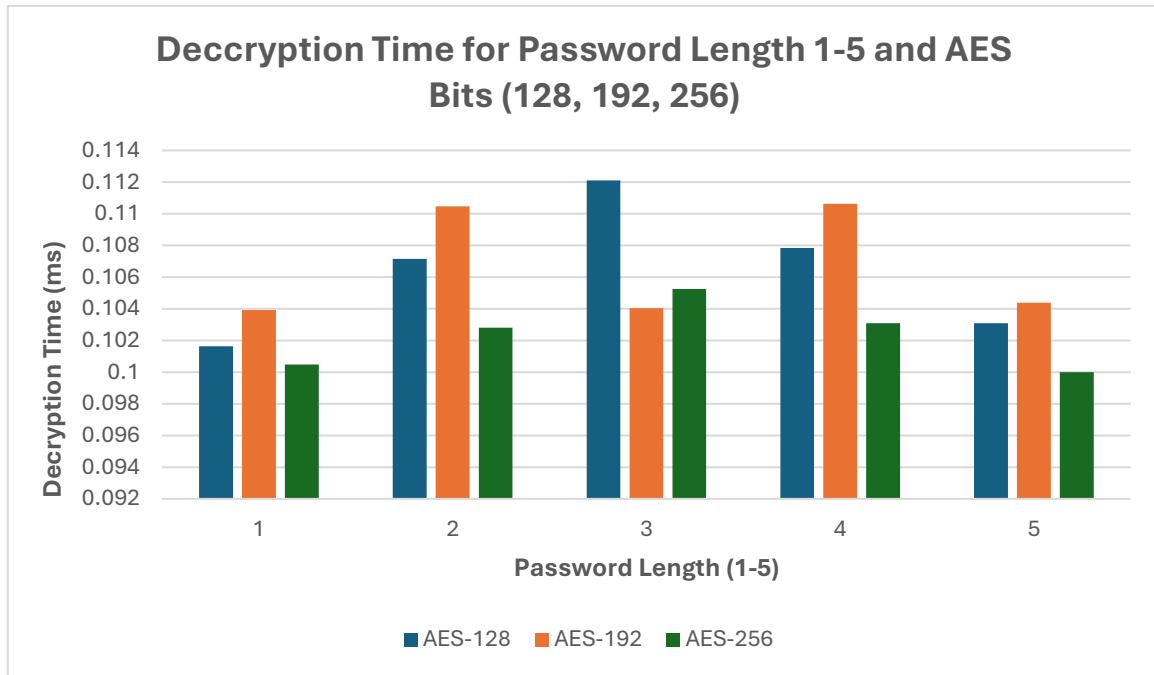


Figure 5.14 Decryption Time for AES Bit Sizes 128, 192, and 256

Note. This figure shows the decryption time for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Decryption time refers to the amount of time it takes for the system to reverse the encryption process, turning the encrypted data back into its original plaintext form. This metric is important in evaluating the performance of encryption systems, as it determines how quickly encrypted data can be accessed and used. In Figure 5.14, the decryption time for AES bit sizes 128, 192, and 256 across password lengths from 1 to 5 shows that AES-256 performs the fastest for password lengths 1, 2, 4, and 5, while AES-128 achieves the fastest decryption time for password length 3. The superior decryption performance of AES-256 for most password lengths can be attributed to its optimization for stronger encryption processes, which can be more efficient in terms of decryption speed.

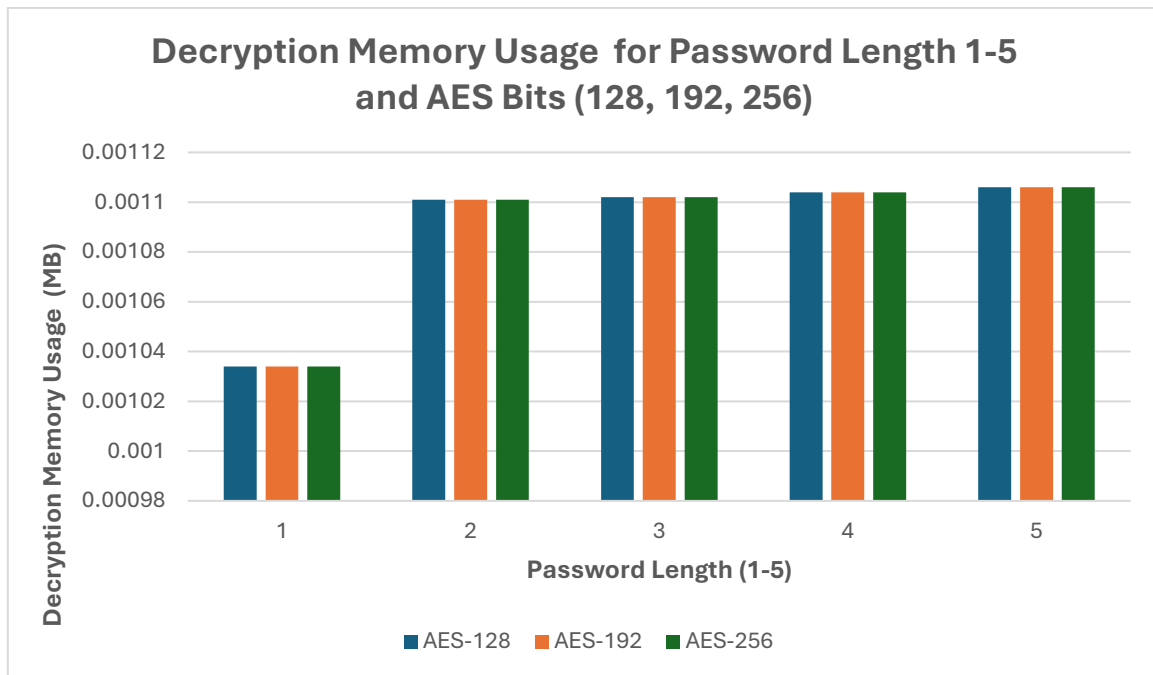


Figure 5.15 Decryption Memory Usage for AES Bit Sizes 128, 192, and 256

Note. This figure shows the decryption memory usage for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Decryption memory usage refers to the amount of memory required by the system to perform the decryption process, which includes memory for decryption keys, buffers, and any temporary data used during decryption. Efficient decryption methods aim to minimize memory consumption while ensuring secure data retrieval. In Figure 5.15, the decryption memory usage is identical across all three AES bit sizes for each password length. This suggests that the choice of AES bit size does not significantly affect decryption memory usage, allowing users to focus on other factors such as encryption speed, security, and overall performance, without concerns about increased memory demands during decryption.

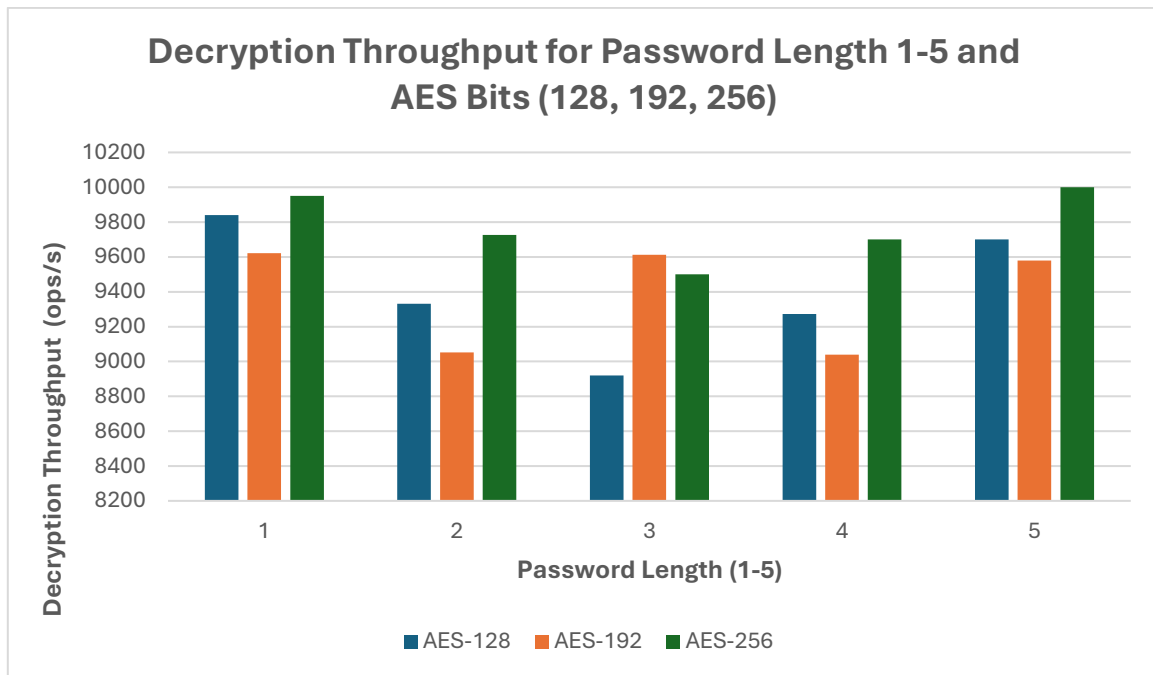


Figure 5.16 Decryption Throughput for AES Bit Sizes 128, 192, and 256

Note. This figure shows the decryption throughput for password lengths ranging from 1 to 5 characters with AES bit sizes of 128, 192, and 256.

Decryption throughput, measured in operations per second (ops/s), indicates how many decryption operations the system can perform within a given time frame. As shown in Figure 5.16, AES-256 consistently outperforms the other bit sizes across most password lengths (1, 2, 4, and 5), demonstrating its efficiency in decryption tasks.

In summary, AES-128 achieves the fastest encryption time and demonstrates superior encryption throughput, making it more efficient for encrypting data, especially for longer passwords. In contrast, AES-256 provides the fastest decryption time and outperforms the other bit sizes in decryption throughput, making it highly effective for decrypting data. Regarding encryption and decryption memory usage, there is no noticeable difference between the AES bit sizes, as all show identical memory consumption. Overall, AES-128 excels in encryption speed, while AES-256 is more efficient for decryption tasks. The optimal choice will depend

on the specific performance requirements, with AES-128 being better suited for encryption and AES-256 for decryption.

5.3.2 Security Testing

Security testing is essential to verify that the Password Management System is resistant to various threats and vulnerabilities. The primary objective was to assess the security of stored passwords and ensure that the system can withstand common attack methods, including brute-force, AES brute-force, dictionary, and rainbow table attacks. To achieve this, the system's built-in password strength tester was utilized to simulate these attacks. For each password, the tester calculates the estimated time to crack and simulates the attack to determine the actual time to crack, providing a comprehensive evaluation of the password's resilience. The results of these simulations are summarized in Table 5.10 for the estimated time to crack and in Table 5.11 for the actual attack time. This testing ensures that the system offers robust protection against potential breaches and allows users to evaluate the security of their passwords in real-world attack scenarios.

Input Used in Password Strength Tester:

- **Test Password:** }M2-83j*=0Of (12 characters, consisting of uppercase, lowercase, digits, and special characters)
- **Guess Per Second:** 3,000,000
- **Thread Count:** 1

- **Password Length 1:** Single character (e.g., "a")
Rationale: This represents the simplest possible password, used for the lower bound of password strength testing.
- **Password Length 2:** Two characters (e.g., "a1")
Rationale: A small increase in length and complexity, combining lowercase letters and digits.
- **Password Length 3:** Three characters (e.g., "a1B")
Rationale: Adds uppercase letters to increase complexity, making the password more secure.
- **Password Length 4:** Four characters (e.g., "a1B*")
Rationale: Introduces a special character to the password, further increasing its strength.
- **Password Length 5:** Five characters (e.g., "a1B*9")
Rationale: The length and complexity increase further, using lowercase, uppercase, digits, and special characters to simulate stronger passwords.

Table 5.10 Estimated Time to Crack

Password Length Attack Type	1	2	3	4	5
Password Brute Force	0.0000086667 seconds	0.000432 seconds	0.0794426667 seconds	26.02497 seconds	2446.347 seconds
AES Brute Force	AES-128: 3.60×10^{24} years	AES-128: 3.60×10^{24} years	AES-128: 3.60×10^{24} years	AES-128: 3.60×10^{24} years	AES-128: 3.60×10^{24} years

	AES-192: 6.63×10^{43} years	AES-192: 6.63×10^{43} years	AES-192: 6.63×10^{43} years	AES-192: 6.63×10^{43} years	AES-192: 6.63×10^{43} years
	AES-256: 1.22×10^{63} years	AES-256: 1.22×10^{63} years	AES-256: 1.22×10^{63} years	AES-256: 1.22×10^{63} years	AES-256: 1.22×10^{63} years
Dictionary Attack	19.09691 seconds	19.09691 seconds	19.09691 seconds	19.09691 seconds	19.09691 seconds
Rainbow Table Attack	323.77 seconds	388.49 seconds	237.52 seconds	268.34 seconds	1748.55 seconds

Note. This table outlines the estimated time to crack for each password length, based on various attack methods.

Based on Table 5.10, the estimated time to crack increases as the password length grows for all attack methods, except for AES brute force. For password brute force, the time grows significantly with each additional character, from 0.0000086667 seconds for a single-character password to 2446.347 seconds for a five-character password. For AES brute force, the estimated time to crack remains consistent across all password lengths, but it increases with larger AES bit sizes, indicating that AES encryption provides robust protection regardless of the password length, with impractically long crack times (measured in millions of years). The dictionary attack shows a constant estimated time to crack of 19.09691 seconds for all password lengths, which implies that the effectiveness of dictionary-based attacks is not influenced by the length of the password. Lastly, for rainbow table attacks, the estimated time to crack increases with password length, ranging from 323.77 seconds for a one-character password to 1748.55 seconds for a five-character password, suggesting that longer passwords are more resistant to rainbow

table attacks. Overall, password brute-force and rainbow table attacks show a clear dependence on password length, while AES brute force and dictionary attacks are less sensitive to length variations.

Table 5.11 Actual Attack Time

Password Length Types of Attacks	1	2	3	4	5
Password Brute Force	0 seconds	0 seconds	0 seconds	0.40 seconds	40.61 seconds
AES Brute Force	N/A	N/A	N/A	N/A	N/A
Dictionary Attack	0.42 seconds	0.32 seconds	11.72 seconds	11.53 seconds	11.6 seconds
Rainbow Table Attack	323.77 seconds	388.49 seconds	237.52 seconds	268.34 seconds	1748.55 seconds

Note. This table shows the actual time taken to complete the attack for each password length, based on various attack methods.

Based on Table 5.11, the actual time to crack increases as the password length grows for most attack methods. For password brute force, the time remains negligible for passwords of lengths 1 to 3, increasing to 0.4 seconds for length 4, and significantly rising to 40.61 seconds for length 5, showing a clear increase in cracking time with longer passwords. The AES brute force method is not applicable as it would take millions of years to complete, making it computationally infeasible to finish. For the dictionary attack, the cracking time starts at 0.42 seconds for a one-character password and slightly decreases to 0.32 seconds for a two-character

password. The time then increases to 11.72 seconds for a three-character password and remains fairly stable at around 11.5 seconds for passwords of length 4 and 5, indicating that the dictionary attack's time becomes less sensitive to password length beyond a certain threshold. Finally, for the rainbow table attack, the time increases significantly with password length, from 323.77 seconds for a one-character password to 1748.55 seconds for a five-character password, indicating that longer passwords are more resistant to rainbow table attacks. Overall, password brute force and rainbow table attack times increase with password length, while dictionary attack times show a more moderate increase after a certain length.

To address the four common password attacks as mentioned above, the developed Password Management System offers robust protection through a combination of security features. For password brute force attacks, the system minimizes risks by implementing a password generation tool that creates strong, complex passwords, making brute-force attempts less feasible. Additionally, the system includes a password breach check that alerts users if their passwords appear in known data breaches, further reducing the likelihood of exposure. To protect against brute-force attacks, the system limits failed login attempts to five, after which access is locked for one hour, either using the master password or recovery key. The dictionary attack is mitigated by ensuring that generated passwords are highly complex and by using the same failed login attempt mechanism to prevent repeated guesses. For rainbow table attacks, the system uses a dynamic salt mechanism in the PBKDF2 key generation process, combined with AES encryption, to secure passwords. Furthermore, the system adds an extra layer of security by encrypting the database using the pysqlitecipher module, which employs SHA-512 to further protect the stored data. These combined features ensure that the system provides robust protection against a wide range of attacks while enabling users to assess and improve their password security.

5.4 Summary

This chapter detailed the various testing phases conducted on the Password Management System, including functional testing (unit testing and user acceptance testing) and non-functional testing (performance and security testing). Each type of test helped identify potential issues and verify that the system met the specified requirements for security, performance, and usability. Through these tests, the system was validated to function as expected, offering a secure and user-friendly password management solution.

CHAPTER 6: CONCLUSION AND FUTURE WORKS

6.1 Overview

This chapter marks the conclusion of the development of the Password Management System, where the objectives, achievements, limitations, and possible future improvements are summarised. The system has undergone rigorous testing and has demonstrated its ability to meet the defined goals. However, like any project, there are always areas for improvement and additional functionalities that can be explored in future iterations.

6.2 Achievements

The Password Management System has successfully met its primary objectives, as summarized in Table 6.1. The system offers enhanced password security, mitigating the risk of weak, reused, and compromised passwords by incorporating a built-in password generator and a breach check function using the Pwned Passwords API¹. It also addresses the issue of single points of failure by incorporating MFA and recovery keys for secure vault access. Furthermore, the system provides transparent password strength evaluation through simulated attacks, allowing users to test their passwords against various attack methods to better understand their password security.

Table 6.1 Objectives and Achievements

Objectives	Achievements
Enhancing Password Security: To tackle the issue of weak, reused, and compromised passwords, the system will include a built-in password generator for creating strong, unique passwords and a breach check function that uses the Pwned Passwords API¹ to alert users if their passwords have appeared in known data breaches.	The system includes a password generator for creating strong passwords and a breach check function using the Pwned Passwords API.
Mitigating Single Point of Failure: To resolve the vulnerability posed by the reliance on a master password, the system will incorporate multi-factor authentication (MFA) and recovery keys, enabling secure recovery of the password vault even if the master password is forgotten or compromised.	MFA and recovery keys are implemented to ensure secure recovery of the password vault even if the master password is compromised.
Providing Transparent Password Strength Evaluation Through Simulated Attacks: To help users understand the actual strength of their passwords, the system will	Simulated attacks, including brute force, AES brute-force, dictionary, and rainbow table attacks, are integrated into the system,

incorporate a feature that simulates multiple attack methods, including plaintext password brute-force, AES brute-force, dictionary attacks, and rainbow table attacks. For each method, the system will provide both an estimated time to crack the password and a real-time simulation of the attack process. This will provide users with a comprehensive understanding of their password's resilience and offer actionable feedback for improvement.	providing users with both estimated and real-time feedback on their password's strength.
---	---

Note. This table outlines the objectives and achievements of the Password Management System, highlighting key features such as password security enhancements, mitigation of single points of failure through MFA and recovery keys, and providing transparent password strength evaluation via simulated attacks.

6.3 Limitations and Constraints

While the Password Management System successfully meets the intended objectives, there are some limitations and constraints that need to be addressed:

1. **System Dependencies:** The system's effectiveness is reliant on internet access for breach checking via the Pwned Passwords API, potentially limiting its functionality in offline environments.

2. **Password Strength Evaluation Scope:** The simulated attacks, while comprehensive, may not cover all possible real-world attack vectors, and additional methods could be incorporated in future versions.
3. **MFA Configuration Complexity:** While MFA improves security, it introduces complexity for users who may not be familiar with authentication methods beyond a single password, suggesting a need for better user guidance.

6.4 Future Works

As with any software, there are always opportunities for further development and refinement.

Here are several potential enhancements that could be implemented in the future:

1. **Additional Password Attack Simulations:** Introducing more attack methods, such as keylogging or phishing simulations, to provide an even more comprehensive evaluation of password strength.
2. **Mobile Version of the System:** Developing a mobile version of the password manager to allow users to manage passwords securely on-the-go.
3. **Offline Breach Checking:** Enhancing the system to allow offline breach checking by integrating local databases of common password leaks.
4. **Further User Education:** Developing tutorials or more in-depth explanations for password security and MFA to improve user understanding of the system's security measures.
5. **AI-Based Password Strength Enhancements:** Exploring the use of AI or machine learning models to improve password generation or even predict potential breaches based on emerging trends.

6. **Improve Usability:** Refining the user interface (UI) and enhancing UX (user experience) to make the system even more intuitive and accessible, especially for less tech-savvy users.

6.5 Summary

Chapter 6 concludes the project by summarizing the achievements, highlighting the objectives that were successfully met, and discussing the limitations and constraints encountered during the development process. While the Password Management System fulfills its intended purpose of offering secure password management, the system is not without room for improvement. The chapter also emphasizes that no system is perfect and outlines possible future enhancements that could improve the system's security, functionality, and user experience.

REFERENCES

- 6sense. (2024). *Keeper Password Manager*. Retrieved from 6sense: <https://6sense.com/tech/password-management/keeper-password-manager-market-share>
- Abikoye, O. C., Haruna, A. D., Abubakar, A., Akande, N. O., & Asani, E. O. (2019). Modified Advanced Encryption Standard Algorithm. *Symmetry*, 11(12), 1484. <https://doi.org/10.3390/sym11121484>
- AgileBits. (n.d.). *1Password*. Retrieved from 1Password: <https://1password.com/>
- Alenezi, M. N., Alabdulrazzaq, H. K., & Mohammad, N. Q. (2020). Symmetric Encryption Algorithms: Review and Evaluation Study. *International Journal of Communication Networks and Information Security (IJCNIS)*, 12(2), 256-272. <https://doi.org/10.17762/ijcnis.v12i2.4698>
- Ali, H. J., Jawad, T. M., & Zuhair, H. (2021). Data Security Using Random Dynamic Salting and AES Based on Master-Slave Keys for Iraqi Dam Management System. *Indonesian Journal of Electrical Engineering and Computer Science*, 23(2), 1018-1029. <https://doi.org/10.11591/ijeecs.v23.i2.pp1018-1029>
- Archana, B., & Niranjana, V. (2023). Overview of Cryptography. *Data Analytics and Artificial Intelligence*, 3(2), 74-80. <https://doi.org/10.46632/daai/3/2/15>
- Atzori, M., Calò, E., Caruccio, L., Cirillo, S., Polese, G., & Solimando, G. (2024). Evaluating Password Strength Based on Information Spread on Social Networks: A Combined Approach Relying on Data Reconstruction and Generative Models. *Online Social Networks and Media*, 42(10), 1-27. <https://doi.org/10.1016/j.osnem.2024.100278>

- Ayyagari, R., Lim, J., & Hoxha, O. (2019). Why Do Not We Use Password Managers? A Study on the Intention to Use Password Managers. *Contemporary Management Research*, 15(4), 227–245. <https://doi.org/10.7903/cmr.19394>
- Baskar, K., Muthumanickam, K., Vijayalakshmi, P., & Kumarganesh, S. (2024). A Strong Password Manager Using Multiple Encryption Techniques. *Journal of the Institution of Engineers (India) Series B*. <https://doi.org/10.1007/s40031-024-01144-6>
- Choi, H., & Seo, S. C. (2021). Optimization of PBKDF2 Using HMAC-SHA2 and HMAC-LSH Families in CPU Environment. *IEEE Access*, 9, 40165–40177. <https://doi.org/10.1109/access.2021.3065082>
- Choi, H., & Seo, S. C. (2021). Optimization of PBKDF2 Using HMAC-SHA2 and HMAC-LSH Families in CPU Environment. *IEEE Access*, 9, 40165 - 40177. <https://doi.org/10.1109/access.2021.3065082>
- Fernando, P., Karunatilake, C., & Liyanage, C. (2023). Challenges and Opportunities in Password Management: A Review of Current Solutions. *Sri Lanka Journal of Social Sciences and Humanities*, 3(2), 9–20. <https://doi.org/10.4038/sljssh.v3i2.96>
- John, A. L. (2024). *Sticky Password*. Retrieved from PasswordHero: <https://passwordhero.com/reviews/sticky-password/#sticky-password-overview>
- Judge, R. (2022). *Bitwarden Releases Global Password Survey in Advance of World Password Day*. Retrieved from Business Wire: <https://www.businesswire.com/news/home/20220426005509/en/Bitwarden-Releases-Global-Password-Survey-in-Advance-of-World-Password-Day>

- Keeper Security. (n.d.). *Keeper*. Retrieved from Keeper Security:
<https://www.keepersecurity.com/>
- Khande, R., Ramaswami, S., Naidu, C., & Patel, N. (2021). An Effective Mechanism for Securing and Managing Password using AES-256 Encryption & PBKDF2. *International Journal of Electrical Engineering & Technology*, 12(5), 1–7.
<https://doi.org/10.34218/IJEET.12.5.2021.001>
- Lamatine Software. (n.d.). *Stick Password*. Retrieved from Stick Password:
<https://www.stickypassword.com/>
- Liu, H. (2024). Employing Cryptographic Techniques for Data Security and Privacy Preservation. *Transactions on Computer Science and Intelligent Systems Research*, 5, 35-39. <https://doi.org/10.62051/58a35334>
- Master, A. (2022). Password Managers: Secure Passwords the Easy Way. *CERIAS Technical Reports*, 2. <https://doi.org/10.5703/1288284317618>
- Menezes, A., & Stebila, D. (2021). The Advanced Encryption Standard: 20 Years Later. *IEEE Security & Privacy*, 19(6), 98–102. <https://doi.org/10.1109/msec.2021.3107078>
- Microsoft. (n.d.). *Windows 11*. Retrieved from Microsoft: <https://www.microsoft.com/en-us/windows?r=1>
- Mustafa, N. A. (2024). Analysis Attackers' Methods with Hashing Secure Password using CSPRNG and PBKDF2. *Wasit Journal of Engineering Sciences*, 12(2), 60-70.
<https://doi.org/10.31185/ejuow.Vol12.Iss2.502>
- Nurhandhi, M., & Suhendar, A. (2023). Improving Firebase BaaS Service Security in Counseling Chat Applications: AES-256 and CBC Approach for End-to-End Encryption.

- JISA (Jurnal Informatika Dan Sains)*, 6(2), 153–160.
<https://doi.org/10.31326/jisa.v6i2.1783>
- Oguejiofor, K. (2024). *Total Password Review 2024: Is It Any Good?* Retrieved from Safety Detectives: <https://www.safetymdetectives.com/best-password-managers/totalpassword/>
- Pargaonkar, S. (2023). A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering. *International Journal of Scientific and Research Publications*, 13(8), 120–124.
<https://doi.org/10.29322/ijsrp.13.08.2023.p14015>
- Preez, P. (2024). *AES256-GCM — Key Rotation in C#*. Retrieved from Medium: <https://medium.com/@pierrephilip/aes256-gcm-key-rotation-in-c-2be80c03cac2>
- Python Software Foundation. (2001). *Python*. Retrieved from Python.org: <https://www.python.org/about/>
- Rahalkar, C., & Gujar, D. (2019). A Secure Password Manager. *International Journal of Computer Applications*, 178(44), 5–9. <https://doi.org/10.5120/ijca2019919323>
- Sarkar, B., Saha, A., Dutta, D., Sarkar, G. D., & Karmakar, K. (2024). A Survey on the Advanced Encryption Standard (AES): A Pillar of Modern Cryptography. *International Journal of Computer Science and Mobile Computing*, 13(4), 68-87.
<https://doi.org/10.47760/ijcsmc.2024.v13i04.008>
- Smid, M. E. (2021). Development of the Advanced Encryption Standard. *Journal of Research of the National Institute of Standards and Technology*, 126(126024), 1-18.
<https://doi.org/10.6028/jres.126.024>

Soft Perfect. (2024). *Difference between Portable and Installer Software*. Retrieved from Soft

Perfect:

<https://www.softperfect.com/contact/knowledgebase.php?article=29#:~:text=Portable%20application%20is%20a%20standalone,change%20anything%20in%20the%20system.>

SQLite. (n.d.). *What is SQLite?* Retrieved from SQLite: <https://www.sqlite.org/>

Taneski, V., Heričko, M., & Brumen, B. (2019). Systematic Overview of Password Security

Problems. *Acta Polytechnica Hungarica*, 16(3).

<https://doi.org/10.12700/aph.16.3.2019.3.8>

Team SignHouse. (2024). *1Password Revenue and Growth Statistics (2024)*. Retrieved from

SignHouse: <https://usesignhouse.com/blog/1password-stats/?form=MG0AV3>

Total Password. (n.d.). *Total Password*. Retrieved from Total Password:

<https://www.totalpassword.com/>

Visual Studio Code. (n.d.). *Visual Studio Code*. Retrieved from Visual Studio Code:

<https://code.visualstudio.com/>

W3Schools. (n.d.). *Introduction to SQL*. Retrieved from W3Schools:

https://www.w3schools.com/sql/sql_intro.asp

APPENDICES

APPENDIX A: SURVEY ON PASSWORD MANAGEMENT PRACTICES

Section 1: General Information

Please provide the following details:

1. Age

- ☐ Under 18
- ☐ 18-24
- ☐ 25-34
- ☐ 35-44
- ☐ 45-54
- ☐ 55-64
- ☐ 65 and above

2. Gender

- ☐ Male
- ☐ Female

3. Occupation

- ☐ Student

- ☐ Employed/Self-Employed
- ☐ Unemployed
- ☐ Retiree

Section 2: Current Concerns About Password Management Practices

1. How many online accounts do you own?

- ☐ 1-3
- ☐ 4-6
- ☐ 7-10s
- ☐ More than 10

2. How often do you reuse passwords across multiple accounts?

- ☐ Never
- ☐ Sometimes
- ☐ Often

3. Have you or someone you know ever experienced an account being hacked or a password being leaked?

- ☐ Yes

☐ No

4. Do you prefer to use a digital password manager over traditional methods of writing/memorizing passwords?

☐ Yes

☐ No

☐ Maybe

☐ Other (please specify)

**5. What concerns do you have about your current password management practices?
(Select all that apply)**

☐ Remembering passwords

☐ Creating strong passwords

☐ Managing multiple accounts

☐ Security concerns with online password storage

☐ Other (please specify)

Section 3: Preferences for a Password Management System

1. What features would you find most useful in a password management system? (Select 3 or more)

- ☐ Offline accessibility
- ☐ Portability
- ☐ Multi-factor authentication (MFA) or biometrics
- ☐ Recovery keys
- ☐ Real-time alerts
- ☐ Built-in password generator
- ☐ Password health checker
- ☐ Backup and restore functions
- ☐ Brute force protection
- ☐ Other (please specify)

2. How important is it for a password manager to have a built-in password generator that creates strong, unique passwords?

(1 = Not Important, 5 = Very Important)

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

3. How likely are you to use a built-in password generator in a password manager?

(1 = Unlikely, 5 = Very Likely)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

4. How likely are you to use a password manager that works offline, ensuring access even without an internet connection?

(1 = Unlikely, 5 = Very Likely)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

5. How likely are you to use multi-factor authentication (MFA) or other security methods like biometrics or fingerprints for added security?

(1 = Unlikely, 5 = Very Likely)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

6. How important is it to have recovery keys for account recovery?

(1 = Not Important, 5 = Very Important)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

7. How likely are you to adopt a password management system that addresses your security and convenience concerns?

(1 = Unlikely, 5 = Very Likely)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

8. How effective do you believe a password management system would be in enhancing your password security and convenience?

(1 = Not Effective, 5 = Very Effective)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

Section 4: User Experience and Feedback

1. What challenges or concerns might prevent you from using a password management system? (Select all that apply)

- ☐ Concerns about security and privacy
- ☐ Difficulty in setting up and using the system
- ☐ Cost of the password manager
- ☐ Trust issues with the software provider
- ☐ Fear of losing access to all passwords if the system fails
- ☐ Preference for traditional methods (e.g., writing down/memorizing passwords)
- ☐ Lack of awareness or understanding of password managers
- ☐ Other (please specify)

2. What additional features or suggestions would you like to see in a password management system?

(Optional)

APPENDIX B: CODE SNIPPET FOR PASSWORD GENERATION

```
# Function to generate random password
def generate_password(length=12, characters=None):
    if characters is None: # Default character set if none is provided
        characters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(characters) for _ in range(length))

def generate_now(ui_context):
    global sqlite_obj
    """Generate a strong password based on stored criteria and update UI."""
    entry_password = ui_context["entry_password"]
    parent_window = ui_context["parent_window"]

    # Check if password field has content
    current_password = entry_password.get()
    if current_password:
        # Ask for confirmation
        confirm = messagebox.askyesno(
            "Confirm Overwrite",
            "This will overwrite your existing password. Are you sure?",
            parent=parent_window # Use parent window from context
        )
        if not confirm:
            return # User canceled

    # Load password criteria from the database
    try:
        # Fetch the latest password criteria from the database
        result = sqlite_obj.getDataFromTable(
            "password_criteria",
            raiseConversionError=True,
            omitID=True
        )

        if result[1]:
            criteria = result[1][0] # Get the first (and only) row
            length, include_uppercase, include_lowercase, include_digits, include_minus, \
            include_underline, include_space, include_special, include_brackets, include_lat1n1 = criteria[0:]
        else:
            messagebox.showwarning("Criteria Error", "No password generation criteria found.", parent=parent_window)
            return

    except Exception as e:
        messagebox.showerror("Database Error", f"Failed to load password criteria: {e}", parent=parent_window)
        return

    # Build character set
    characters = ""
    if include_uppercase == 'True': characters += string.ascii_uppercase
    if include_lowercase == 'True': characters += string.ascii_lowercase
    if include_digits == 'True': characters += string.digits
    if include_minus == 'True': characters += "-"
    if include_underline == 'True': characters += "_"
    if include_space == 'True': characters += " "
    if include_special == 'True': characters += "!\"#$%&'()*+,-./:;=?@\\^`{|}~"
    if include_brackets == 'True': characters += "[]{}()<>"
    if include_lat1n1 == 'True': characters += ''.join(chr(i) for i in range(160, 256))

    if not characters:
        messagebox.showwarning("Selection Error", "Please select at least one character type.", parent=parent_window)
        return

    max_attempts = 1000 # Limit attempts to avoid infinite loop
    for _ in range(max_attempts):
        new_password = generate_password(int(length), characters)
        if check_password_strength(new_password) == "Strong":
            break
    else:
        messagebox.showerror(
            "Generation Failed",
            "Unable to generate a strong password with the selected criteria.\n\n"
            "To fix this, please update your password criteria with the following tips:\n\n"
            "- Use at least 12 characters\n"
            "- Include uppercase and lowercase letters\n"
            "- Add digits (0-9)\n"
            "- Include special characters (e.g. !@#$%^&*)\n",
            parent=parent_window
        )
        return

    # Insert password and update strength UI
    entry_password.delete(0, tk.END)
    entry_password.insert(0, new_password)

    update_password_strength(ui_context)
```

Note. This appendix illustrates the code snippet for generating a secure password based on user-defined criteria, such as length, uppercase letters, lowercase letters, digits, special characters, spaces, brackets,

and Latin-1 characters. The `generate_now()` function retrieves these criteria from the database, constructs a character set, and generates a password that meets the strength requirements. If the password is strong, it is displayed in the user interface; otherwise, the system prompts the user to adjust the criteria

APPENDIX C: CODE SNIPPET FOR ESTIMATING PASSWORD CRACK TIME

```
def estimate_crack_time(password, aes_bits, method, dictionary_path, rainbow_path, guesses_per_sec):
    """
    - Password Brute-Force
    - AES Brute-Force
    - Dictionary Attack
    - Rainbow Table
    """
    # Early exit on empty password
    if not password:
        return {m: "No password entered" for m in
                ["Password Brute-Force", "AES Brute-Force",
                 "Dictionary Attack", "Rainbow Table"]}

    # Check for invalid guesses_per_sec
    if guesses_per_sec <= 0:
        return {method: "Invalid guess rate: must be greater than 0"}

    # Character set size & entropy
    cs = get_charset_size(password)
    possible_combinations = cs ** len(password)
    total_entropy = len(password) * math.log2(cs)

    results = {}

    # - Password Brute-Force -
    if method == "Password Brute-Force":
        bf_seconds = possible_combinations / guesses_per_sec
        results["Password Brute-Force"] = format_time(bf_seconds)
    else:
        results["Password Brute-Force"] = "Password brute-force disabled"

    # - AES Brute-Force -
    if method == "AES Brute-Force":
        aes_seconds = (2 ** aes_bits) / guesses_per_sec
        results["AES Brute-Force"] = format_time(aes_seconds)
    else:
        results["AES Brute-Force"] = "AES brute-force disabled"

    # - Dictionary Attack -
    if method == "Dictionary Attack":
        dict_result = "Path not found"
        if os.path.exists(dictionary_path):
            size = count_lines(dictionary_path)
            dict_time = estimate_dict_time(size, guesses_per_sec)
            found = password_in_dict(password, dictionary_path)
            if found:
                dict_result = f"{dict_time} (found)"
            else:
                dict_result = "Not found in wordlist"
            results["Dictionary Attack"] = dict_result
        else:
            results["Dictionary Attack"] = "Dictionary attack disabled"
```

```

# - Rainbow Table -
if method == "Rainbow Table":
    # Rainbow table time estimation
    try:
        # Get base_time from rainbow_crack_time
        rainbow_result = sqlite_obj.getDataFromTable(
            "rainbow_crack_time",
            raiseConversionError=True,
            omitID=True
        )

        base_time = None
        if rainbow_result[1]:
            for row in rainbow_result[1]:
                if int(row[0]) == len(password): # First column is length
                    base_time = row[1]          # Second column is base_time
                    break

        if base_time is None:
            results["Rainbow Table"] = "No estimate for this password length"
        else:
            # Get thread_count from attack_settings
            attack_result = sqlite_obj.getDataFromTable(
                "attack_settings",
                raiseConversionError=True,
                omitID=True
            )

            threads = 1
            if attack_result[1]:
                try:
                    # Thread count is 4th column (index 3)
                    threads = max(1, int(attack_result[1][0][3]))
                except (TypeError, ValueError):
                    pass # Keep default value if conversion fails

            adjusted_time = base_time / threads
            crack_time = format_time(adjusted_time)
            results["Rainbow Table"] = crack_time

    except Exception as e:
        results["Rainbow Table"] = f"Database error: {str(e)}"

else:
    results["Rainbow Table"] = "Rainbow attack disabled"

```

Note. This appendix contains the code snippet for estimating the time required to crack a password using four different attack methods: password brute force, AES brute force, dictionary attack, and rainbow table attack. The `estimate_crack_time()` function calculates the estimated cracking time for each method based on the password's entropy and the number of guesses per second (`guesses_per_sec`). The guesses per second can be either user-defined or determined by running a test on the user's device to measure the guess rate, as demonstrated in Figure 4.13.

APPENDIX D: CODE SNIPPET FOR ACTUAL DICTIONARY ATTACK TIME

```

elif method == "Dictionary Attack":
    post(f"Dictionary path: {dict_path}")
    if not os.path.exists(dict_path):
        post("Dictionary path not found")
        return

    # Get all dictionary files
    dict_files = []
    if os.path.isdir(dict_path):
        for root, _, files in os.walk(dict_path):
            for file in files:
                file_path = os.path.join(root, file)
                if os.path.isfile(file_path):
                    dict_files.append(file_path)
        post(f"Found {len(dict_files)} dictionary files in directory")
    else:
        dict_files = [dict_path]

    # Shared resources
    found_event = threading.Event()
    tried_set = set()
    found_info = {"file": "", "line": 0}
    dict_size = [0]
    lock = threading.Lock()

    def dictionary_worker(file_chunk):
        nonlocal found_info, dict_size
        for file_path in file_chunk:
            if found_event.is_set() or stop_flag.is_set():
                return

            try:
                with open(file_path, 'r', errors='ignore') as f:
                    display_path = os.path.relpath(file_path, dict_path) if os.path.isdir(dict_path) else os.path.basename(file_path)
                    with lock:
                        post(f"Scanning dictionary file: {display_path}")

                    for line_num, line in enumerate(f, 1):
                        if found_event.is_set() or stop_flag.is_set():
                            return

                        word = line.strip()
                        if not word:
                            continue

                        with lock:
                            dict_size[0] += 1
                            tried_set.add(word)

                        if word == password:
                            with lock:
                                found_info["file"] = os.path.relpath(file_path, dict_path) if os.path.isdir(dict_path) else os.path.basename(file_path)
                                found_info["line"] = line_num
                                found_event.set()
                            return
            except Exception as e:
                with lock:
                    post(f"Error reading {os.path.basename(file_path)}: {str(e)}")

    # Split files into chunks for threads
    chunk_size = max(1, len(dict_files) // threads)
    file_chunks = [dict_files[i:i+chunk_size] for i in range(0, len(dict_files), chunk_size)]

    workers = []
    for chunk in file_chunks:
        t = threading.Thread(target=dictionary_worker, args=(chunk,), daemon=True)
        workers.append(t)
        t.start()

    # Wait for threads to complete or password found
    for t in workers:
        t.join()

    attempt_counter[0] += dict_size[0]

    if found_event.is_set():
        post(f"Password found in dictionary: {found_info['file']} (Line {found_info['line']})")
    else:
        if not stop_flag.is_set():
            post(f"Password not found in any dictionary file (searched {dict_size[0]} words)")
            post("Proceeding with brute-force excluding tried words...")

            # Brute-force but skip tried_set
            charset = string.ascii_letters + string.digits + string.punctuation
            found_event = threading.Event()

            def filtered_brute_force_worker(start_chars, charset, target, found_event, attempt_counter, stop_flag):
                max_len = len(target) + 1
                for length in range(1, max_len + 1):
                    for first in start_chars:
                        for combo in itertools.product(charset, repeat=length - 1):
                            if found_event.is_set() or stop_flag.is_set():
                                return
                            attempt = first + ''.join(combo)
                            if attempt in tried_set:
                                continue # Skip already tried words
                            attempt_counter[0] += 1
                            if attempt == target:
                                found_event.set()
                                post(f"Password found by brute-force: {attempt}")
                                return

            workers = []
            for i in range(threads):
                start_chars = charset[i::threads]
                t = threading.Thread(
                    target=filtered_brute_force_worker,
                    args=(start_chars, charset, password, found_event, attempt_counter, stop_flag),
                    daemon=True
                )
                workers.append(t)
                t.start()

            for t in workers:
                t.join()

            if not found_event.is_set() and not stop_flag.is_set():
                post("Password not found even after brute-force")

```

Note. This appendix presents the code snippet for performing a dictionary attack using the `simulate_attack()` function. The function attempts to crack the password by checking it against entries in a dictionary file, as described in Dictionary Attack (Chapter 4.2.5.8.1). It calculates the actual time taken to find a match by iterating through the dictionary, which contains a list of common passwords. If the password is found, the time is reported. If not, the attack automatically proceeds to a password brute force attack.

APPENDIX E: CODE SNIPPET FOR ACTUAL RAINBOW TABLE ATTACK TIME

```
else: # Rainbow Table
    try:
        # Generate MD5 hash of the password
        post("🔧 Generating MD5 hash...")
        md5_hash = hashlib.md5(password.encode()).hexdigest()
        post(f"🔑 MD5 hash: {md5_hash}")

        # Validate paths
        if not os.path.exists(rainbow_path):
            post("❌ Rainbow table path not found")
            return

        # Prepare rcracki_mt command
        command = f"rcracki_mt -h {md5_hash} -t {threads} {rainbow_path}"
        post(f"🚀 Executing: {command}")

        # Execute command in the Crack_Rainbow_Table directory
        crack_dir = "Crack_Rainbow_Table"
        if not os.path.exists(crack_dir):
            post(f"❌ Directory not found: {crack_dir}")
            return

        # With this cross-platform solution:
        if os.name == 'nt': # Windows
            process = subprocess.Popen(
                command,
                cwd=crack_dir,
                shell=True,
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
                text=True,
                creationflags=subprocess.CREATE_NEW_PROCESS_GROUP
            )
        else: # Unix-based systems
            process = subprocess.Popen(
                command,
                cwd=crack_dir,
                shell=True,
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
                text=True,
                preexec_fn=os.setsid
            )
```

```

# Read output in real-time
while True:
    if stop_flag.is_set():
        post("🛑 Stopping rainbow table attack...")
        if os.name == 'nt':
            os.kill(process.pid, signal.CTRL_BREAK_EVENT)
        else:
            os.killpg(os.getpgid(process.pid), signal.SIGTERM)
        break

    output = process.stdout.readline()
    if output == '' and process.poll() is not None:
        break
    if output:
        post(output.strip())

    time.sleep(0.1) # Prevent UI freeze

# Check final result
return_code = process.poll()
if not stop_flag.is_set(): # Only check if not stopped by user
    if return_code == 0:
        post("✅ Password found in rainbow table!")
    else:
        post("❌ Password not found in rainbow tables")
else:
    post("🛑 All Processes Terminated")

except Exception as e:
    post(f"⚠️ Error during rainbow table attack: {str(e)}")

elapsed_time = time.time() - start_time
if not stop_flag.is_set():
    current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    post(f"🕒 End time: {current_time}")
    post(f"🔢 Total attempts: {attempt_counter[0]:,}")
    post(f"⌚ Time taken: {elapsed_time:.2f} seconds")
    post("🏁 Attack completed")
post("")

```

Note. This appendix shows the code snippet for performing a rainbow table attack on a password hash using the `simulate_attack()` function. The function utilizes precomputed hashes from a rainbow table to attempt to crack the password and calculates the actual time taken to find a match, leveraging the `rcracki_mt` tool, as described in Chapter 4.2.5.8.1. The function generates the MD5 hash of the entered password and executes the `rcracki_mt -h {md5_hash} -t {threads} {rainbow_path}` command to initiate the cracking process, utilizing multithreading for efficient execution.

APPENDIX F: AES BIT PERFORMANCE METRICS

```
def test_aes_performance(bits, password, test_enc, test_dec, test_mem, test_throughput, test_entropy):
    results = {}
    plaintext = os.urandom(1 * 1024 * 1024) # 1MB random data
    key_length = bits // 8

    total_enc_time = 0
    total_dec_time = 0
    total_enc_memory = 0
    total_dec_memory = 0
    total_enc_throughput = 0
    total_dec_throughput = 0
    total_enc_entropy = 0
    total_dec_entropy = 0

    try:
        target_key = password.encode().ljust(key_length, b'\0')[:key_length]

        for _ in range(10): # Run the test 10 times
            # Encryption metrics
            if test_enc:
                start_time = time.perf_counter()
                if test_mem:
                    tracemalloc.start()

                iv = os.urandom(12)
                cipher = Cipher(algorithms.AES(target_key), modes.GCM(iv), backend=default_backend())
                encryptor = cipher.encryptor()
                ciphertext = encryptor.update(plaintext) + encryptor.finalize()

                if test_mem:
                    _, peak = tracemalloc.get_traced_memory()
                    tracemalloc.stop()
                    total_enc_memory += peak / (1024 ** 2) # MB

                end_time = time.perf_counter()
                total_enc_time += (end_time - start_time) * 1000 # ms

                if test_throughput:
                    total_enc_throughput += (1 * 1024 * 1024) / (end_time - start_time) / (1024 ** 2) # MB/s

                if test_entropy:
                    total_enc_entropy += calculate_entropy(ciphertext)

            # Decryption metrics
            if test_dec:
                iv = os.urandom(12)
                cipher = Cipher(algorithms.AES(target_key), modes.GCM(iv), backend=default_backend())
                encryptor = cipher.encryptor()
                ciphertext = encryptor.update(plaintext) + encryptor.finalize()
                tag = encryptor.tag
                combined = iv + tag + ciphertext

                start_time = time.perf_counter()
                if test_mem:
                    tracemalloc.start()

                iv = combined[:12]
                tag = combined[12:28]
                ciphertext_part = combined[28:]

                cipher = Cipher(algorithms.AES(target_key), modes.GCM(iv, tag), backend=default_backend())
                decryptor = cipher.decryptor()
                decrypted = decryptor.update(ciphertext_part) + decryptor.finalize()

                if test_mem:
                    _, peak = tracemalloc.get_traced_memory()
                    tracemalloc.stop()
                    total_dec_memory += peak / (1024 ** 2) # MB

                end_time = time.perf_counter()
                total_dec_time += (end_time - start_time) * 1000 # ms

                if test_throughput:
                    total_dec_throughput += (1 * 1024 * 1024) / (end_time - start_time) / (1024 ** 2) # MB/s

            # Calculate average results
            if test_enc:
                results['encryption_time'] = total_enc_time / 10
                results['encryption_memory'] = total_enc_memory / 10
                results['encryption_throughput'] = total_enc_throughput / 10
                results['encryption_entropy'] = total_enc_entropy / 10 if test_entropy else 0

            if test_dec:
                results['decryption_time'] = total_dec_time / 10
                results['decryption_memory'] = total_dec_memory / 10
                results['decryption_throughput'] = total_dec_throughput / 10

    except Exception as e:
        print(f"Performance test error: {str(e)}")
        return {}

    return results
```

Note. This appendix provides the code snippet for evaluating the performance of different AES bit sizes (128, 192, and 256). The test measures key performance indicators such as encryption time, decryption time, memory usage, and throughput for each bit size. These performance metrics allow users to assess the computational efficiency and resource usage associated with each AES encryption option. The

results help users select the AES bit size that strikes the optimal balance between performance and security for their system.

APPENDIX G: AES BIT SECURITY METRICS

```
def calculate_entropy(data):
    """Calculate Shannon entropy of given data"""
    if not data:
        return 0.0
    entropy = 0.0
    data_size = len(data)
    counts = Counter(data)
    for count in counts.values():
        p_x = count / data_size
        entropy += -p_x * math.log2(p_x)
    return entropy

def estimate_crack_times_evaluation(password, aes_bits, aes, guesses_per_sec):
    """
    Returns a dictionary with estimated crack times for different methods.
    Each method will be enabled/disabled based on the user's selection in the GUI.
    """
    # Early exit if the password is empty
    if not password:
        return {method: "No password entered" for method in
                ["AES Brute-Force"]}

    # Check for invalid guesses_per_sec
    if guesses_per_sec <= 0:
        return {"Error": "Invalid guess rate: must be greater than 0"}

    # Charset size & entropy for password brute-force
    cs = get_charset_size(password) # This function calculates the charset size based on the password
    possible_combinations = cs ** len(password)
    total_entropy = len(password) * math.log2(cs)

    results = {}

    # AES Brute-Force
    if aes:
        aes_seconds = (2 ** aes_bits) / guesses_per_sec # Time to brute-force AES key
        results["AES Brute-Force"] = format_time(aes_seconds)
    else:
        results["AES Brute-Force"] = "AES brute-force disabled"
    return results
```

Note. This appendix illustrates the code snippet for evaluating the security of each AES bit size (128, 192, and 256). The key metrics assessed include ciphertext entropy and the estimated crack time for each encryption method, as mentioned in Section 4.2.5.8.1. These metrics provide insight into the strength and resilience of each AES bit size against potential attacks, helping users choose the most secure option for their password encryption needs.

APPENDIX H: DETERMINING THE BEST AES BIT SIZE

```
def determine_best_aes(results, cli_output):
    # Initialize scores for each AES bit
    scores = {"128": 0, "192": 0, "256": 0}
    total_metrics = {"128": 0, "192": 0, "256": 0} # Tracks the number of metrics counted

    # Store detailed breakdown of scores for display
    score_details = {
        "128": {'enc_time': 0, 'mem_usage': 0, 'throughput': 0, 'entropy': 0, 'crack_time': 0, 'dec_time': 0, 'dec_mem_usage': 0, 'dec_throughput': 0},
        "192": {'enc_time': 0, 'mem_usage': 0, 'throughput': 0, 'entropy': 0, 'crack_time': 0, 'dec_time': 0, 'dec_mem_usage': 0, 'dec_throughput': 0},
        "256": {'enc_time': 0, 'mem_usage': 0, 'throughput': 0, 'entropy': 0, 'crack_time': 0, 'dec_time': 0, 'dec_mem_usage': 0, 'dec_throughput': 0}
    }

    # Maximum score per metric
    max_score_per_metric = 10

    for bits in ["128", "192", "256"]:
        perf = results[bits]['performance']
        est_times = results[bits]['estimates']

        # Calculate score for Encryption Time: Lower is better
        enc_time = perf.get('encryption_time', None)
        if enc_time is not None:
            enc_time_score = max_score_per_metric / (enc_time + 1)
            scores[bits] += enc_time_score
            score_details[bits]['enc_time'] = enc_time_score
            total_metrics[bits] += 1

        # Calculate score for Memory Usage (Encryption): Lower is better
        mem_usage = perf.get('encryption_memory', None)
        if mem_usage is not None:
            mem_usage_score = max_score_per_metric / (mem_usage + 1)
            scores[bits] += mem_usage_score
            score_details[bits]['mem_usage'] = mem_usage_score
            total_metrics[bits] += 1

        # Calculate score for Throughput (Encryption): Higher is better
        throughput = perf.get('encryption_throughput', None)
        if throughput is not None:
            throughput_score = min(throughput, max_score_per_metric)
            scores[bits] += throughput_score
            score_details[bits]['throughput'] = throughput_score
            total_metrics[bits] += 1

        # Calculate score for Entropy: Higher is better
        entropy = perf.get('encryption_entropy', None)
        if entropy is not None:
            entropy_score = min(entropy, max_score_per_metric)
            scores[bits] += entropy_score
            score_details[bits]['entropy'] = entropy_score
            total_metrics[bits] += 1

        # Calculate score for Estimated Crack Time: Lower is better
        crack_time = est_times.get("AES Brute-Force", "N/A")
        if crack_time != "N/A":
            time_in_seconds = convert_crack_time_to_seconds(crack_time)
            crack_time_score = max_score_per_metric / (time_in_seconds + 1)
            scores[bits] += crack_time_score
            score_details[bits]['crack_time'] = crack_time_score
            total_metrics[bits] += 1

        # Calculate Decryption Metrics: Now including decryption time, memory usage, and throughput
        # Score for Decryption Time: Lower is better
        dec_time = perf.get('decryption_time', None)
        if dec_time is not None:
            dec_time_score = max_score_per_metric / (dec_time + 1)
            scores[bits] += dec_time_score
            score_details[bits]['dec_time'] = dec_time_score
            total_metrics[bits] += 1

        # Score for Decryption Memory Usage: Lower is better
        dec_mem_usage = perf.get('decryption_memory', None)
        if dec_mem_usage is not None:
            dec_mem_usage_score = max_score_per_metric / (dec_mem_usage + 1)
            scores[bits] += dec_mem_usage_score
            score_details[bits]['dec_mem_usage'] = dec_mem_usage_score
            total_metrics[bits] += 1

        # Score for Decryption Throughput: Higher is better
        dec_throughput = perf.get('decryption_throughput', None)
        if dec_throughput is not None:
            dec_throughput_score = min(dec_throughput, max_score_per_metric)
            scores[bits] += dec_throughput_score
            score_details[bits]['dec_throughput'] = dec_throughput_score
            total_metrics[bits] += 1

    # Normalize the score to be out of 100, based on the number of metrics available
    for bits in ["128", "192", "256"]:
        if total_metrics[bits] > 0:
            scores[bits] = (scores[bits] / (total_metrics[bits] * max_score_per_metric)) * 100 # Normalize out of 100

    # Select the AES bit with the highest score
    best_bit = max(scores, key=scores.get)

    # Output the detailed scores for each AES bit using cli_output
    cli_output.insert(tk.END, "\n\nDetailed Scores for each AES bit:")
    for bits in ["128", "192", "256"]:
        cli_output.insert(tk.END, f"\n\nAES-{bits}:")
        cli_output.insert(tk.END, f"\n    Encryption Time: {score_details[bits]['enc_time']:.2f}")
        cli_output.insert(tk.END, f"\n    Encryption Memory Usage: {score_details[bits]['mem_usage']:.2f}")
        cli_output.insert(tk.END, f"\n    Encryption Throughput: {score_details[bits]['throughput']:.2f}")
        cli_output.insert(tk.END, f"\n    Decryption Time: {score_details[bits]['dec_time']:.2f}")
        cli_output.insert(tk.END, f"\n    Decryption Memory Usage: {score_details[bits]['dec_mem_usage']:.2f}")
        cli_output.insert(tk.END, f"\n    Decryption Throughput: {score_details[bits]['dec_throughput']:.2f}")
        cli_output.insert(tk.END, f"\n    Ciphertext Entropy: {score_details[bits]['entropy']:.2f}")
        cli_output.insert(tk.END, f"\n    Estimated Crack Time: {score_details[bits]['crack_time']:.2f}")

    # Display the total score out of 100%
    cli_output.insert(tk.END, f"\n    Total Score: {scores[bits]:.2f}%\n")

    cli_output.insert(tk.END, f"\n\nBest AES bit: AES-{best_bit} with a total score of {scores[best_bit]:.2f}%")

    return best_bit
```

Note. This appendix demonstrates the code snippet for determining the optimal AES bit size for a user's device. The function evaluates performance and security metrics for AES-128, AES-192, and AES-256, including encryption/decryption time, memory usage, throughput, entropy, and estimated crack time. Each AES bit size is scored based on these metrics, and the bit size with the highest score is recommended as the best choice for the user's device. This evaluation ensures that users select the most suitable AES bit size to balance security and performance according to their device's capabilities.

APPENDIX I: USER ACCEPTANCE TESTING SURVEY FOR THE DEVELOPED PASSWORD MANAGEMENT SYSTEM

Section 1: Download the Password Management System

Before proceeding with the questions, please download and test the system from the following link. The tutorial can be found in the README section:

https://github.com/Rakoim/FYP_Password_Manager

1. Have you downloaded and tested the Password Management System?

☐ Yes

☐ No

Section 2: User Acceptance Testing Questions

Thank you for confirming that you have downloaded and tested the system. Now, please provide your feedback based on your experience with the system.

1. How easy was it to use the Password Management System?

(1 = Very Difficult, 5 = Very Easy)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

2. Did you encounter any issues during the login process?

☐ Yes

☐ No

3. If yes, please explain:

(Optional)

4. How would you rate the ease of adding or editing password entries?

(1 = Very Difficult, 5 = Very Easy)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

5. Were the password strength and breach check features helpful?

☐ Yes

☐ No

☐ Not Sure

6. Please explain why:

(Optional)

7. How would you rate the AES bit evaluation feature in terms of ease of use?

(1 = Very Difficult, 5 = Very Easy)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

8. How confident are you in the security features of the system (e.g., AES encryption, MFA)?

(1 = Not Confident at All, 5 = Very Confident)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

9. Did you use the "Generate Password" feature? If so, how would you rate it?

☐ Yes

☐ No

10. If yes, please rate it:

(Optional)

(1 = Very Poor, 5 = Excellent)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

11. Was the password strength tester useful for understanding your password's security?

☐ Yes

☐ No

12. Please provide any suggestions to improve this feature:

(Optional)

Section 3: User Experience and Feedback

1. How would you rate the overall user interface and design of the system?

(1 = Very Poor, 5 = Excellent)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

2. What improvements would you suggest for the Password Management System?

(Optional)

3. Overall, how satisfied are you with the Password Management System?

(1 = Very Unsatisfied, 5 = Very Satisfied)

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

4. Additional Comments or Suggestions:

(Optional)
