A

Project Report

On

Secure Passwords with Hidden Information and Non-Exposure

submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

by

Sirikonda Sai Kumar

(20EG105346)

Vangapati Chandan Teja

(20EG105353)

Meda ShyamSunder

(20EG105360)

Under the guidance of

Dr. T. Shyam Prasad

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANURAG UNIVERSITY

VENTAKAPUR (V), GHATKESAR (M), MEDCHAL (D), T.S - 500088

TELANGANA

(2023-2024)

DEPARTMENT OF COMPUTER SCIENCE AND ENGNIEERING

CERTIFICATE

This is to certify that the Project entitled "**Secure Passwords with Hidden Information and Non-Exposure**" being submitted by **Sirikonda Sai Kumar** bearing the Hall Ticket number **20EG105346, Vangapati Chandan Teja** bearing the Hall Ticket number **20EG105353, Meda ShyamSunder** bearing the Hall Ticket number **20EG105360** in partial fulfillment of the requirements for the award of the **Bachelor of Technology** in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision for the academic year 2023 to 2024.

The results embodied in this Project have been verified and found to be satisfactory. The results embody in the Project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide
Dr. T. Shyam Prasad
Assistant Professor, CSE

Dean, CSE
Dr. G. Vishnu Murthy

External Examiner

DECLARATION

We hereby declare that the Report entitled "**Secure Passwords with Hidden Information and Non-Exposure**" submitted for the award of Bachelor of technology Degree is our original work and the Report has not formed the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma**.**

Place: Anurag University, Hyderabad
Date:

Sirikonda Sai Kumar
20EG105346

Vangapati Chandan Teja
20EG105353

Meda ShyamSunder
20EG105360

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Dr. T. Shyam Prasad, Assistant Professor, Department of Computer Science and Engineering**, Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, **Dean, Department of Computer Science and Engineering,** Anurag University. We also express our deep sense of gratitude to **Dr. V. V. S. S. S. Balaram**, Academic coordinator. **Dr. T. Shyam Prasad,** Assistant Professor**,** Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of my project work. We would like to express special thanks to **Dr. V. Vijaya Kumar, Dean School of Engineering,** Anurag University, for his encouragement and timely Support in our B. Tech program.

Sirikonda Sai Kumar

20EG105346

Vangapati Chandan Teja

20EG105353

Meda ShyamSunder

20EG105360

ABSTRACT

In the realm of password management, the SPHINX system introduces a groundbreaking approach that ensures security even in the event of a compromise. Leveraging the device-enhanced password authenticated key exchange (DE-PAKE) model, SPHINX separates the stored information from the user's master password, preventing any leakage even when the device is under full control. This is achieved through the use of an efficient oblivious pseudo-random function (OPRF) scheme, transforming user-memorized passwords into high-entropy, mutually independent, and random passwords without storing them on the device. Un like existing password managers, SPHINX mandates the registration of these transformed passwords with web services, offering resistance against online guessing attacks. Furthermore, in the face of server compromise or malicious intent, SPHINX provides protection against offline dictionary attacks by storing salted one-way hashes of the randomized passwords on the server. The inclusion of website domains in password computations also enhances resistance to phishing attacks. The implementation of SPHINX involves a unique key for each service, facilitating the mapping of user-memorized passwords to randomized passwords through an oblivious PRF protocol between the device and the client. The cryptographic proofs of security underpinning SPHINX's design ensure robust protection against various forms of attacks.

TABLE OF CONTENTS

1. INTRODUCTION **1**

1.1. OVERVIEW **1**

1.2. PROBLEM STATEMENT **2**

1.3. PROBLEM ILLUSTRATION **3**

2. LITERATURE SURVEY **4**

3. PROPOSED METHOD **7**

3.1. ILLUSTRATION **7**

3.2. AUTHENTICATION AND USER MANAGEMENT **7**

3.2.1. USER REGISTRATION **7**

3.2.2. USER LOGIN **8**

3.2.3. ADMIN PANEL **8**

3.3. PASSWORD MANAGEMENT **9**

3.4. OTP VERIFICATION **11**

3.5. USER INTERFACE **13**

3.6. SECURITY MEASURES **15**

4. DESIGN **17**

4.1. UML DIAGRAMS **17**

4.1.1. USE CASE DIAGRAM **17**

4.1.2. SEQUENCE DIAGRAM **21**

4.1.3. CLASS DIAGRAM **25**

4.1.4. COLLABORATION DIAGRAM **26**

4.1.5. STATE DIAGRAM **27**

4.1.6. ACTIVITY DIAGRAM **28**

4.1.7. COMPONENT DIAGRAM **30**

4.1.8. DEPLOYMENT DIAGRAM **31**

5. IMPLEMENTATION **32**

5.1. MODULES **32**

5.1.1. OVERALL IMPLEMENTATION **32**

5.1.2. USER MODULE **33**

5.1.3. ADMIN MODULE **35**

5.1.4. DEPLOYMENT **37**

5.2. SAMPLE CODE **39**

5.2.1. HOME.HTML **39**

5.2.2. MANAGE.PY **44**

5.2.3. VIEWS.PY **45**

6. EXPERIMENT SCREENSHOTS **49**

6.1. HOME PAGE **49**

6.2. ADDING PASSWORDS **49**

6.3. SERVER PAGE **40**

7. OBSERVATIONS **55**

8. SPHNIX vs Other Password Managers **53**

9. CONCLUSION **54**

10. REFERENCES **55**

LIST OF FIGURES

Figure No.	Figure Name	Page No.
2.1	Existing method	5
3.1.1	SPHINX	7
3.2.1	Authentication and User Management	12
4.1.1.1	Use case Diagram	18
4.1.2.1	Sequence Diagram	22
4.1.3.1	Class Diagram	25
4.1.4.1	Collaboration Diagram	26
4.1.5.1	State Diagram	27
4.1.6.1	Activity Diagram	29
4.1.7.1	Component Diagram	30
4.1.8.1	Deployment Diagram	31
5.1.2.1	Admin module	23
5.1.3.1	User module	24
6.1	Home page	38
6.2	Adding Passwords	38
6.3	Server page	39

LIST OF TABLES

Table No.	Table Name	Page No.
2.1	Literature survey	6
8.1	Comparison between Existing and Proposed Methods	53

1. INTRODUCTION

1.1 Overview

The project described is centered around the concept of Device-Enhanced Password-Authenticated Key Exchange (DE-PAKE), which forms the basis for the SPHINX application. DE-PAKE aims to enhance password security by transforming user-memorable passwords into high-entropy random strings using a secondary device, thereby mitigating vulnerabilities associated with low-entropy passwords. The protocol involves four parties: the user (U), client (C), server (S), and device (D).

The DE-PAKE protocol consists of two phases: initialization and authenticated key exchange. During initialization, the user selects a password from a dictionary and communicates with the device and server to establish necessary states for authentication. In the authenticated key exchange phase, the user interacts with the device, client, and server to authenticate and establish session keys, protecting communication with the server.

The SPHINX instantiation of DE-PAKE assumes the availability of a smartphone during the authentication process, providing improved security properties compared to regular password-only authentication. However, it acknowledges limitations in protecting against client compromise, suggesting future integration with two-factor authentication (TFA) solutions.

The project's contribution over its conference publication includes an extended analytical comparison of SPHINX with other password managers based on security, usability, and deployability metrics. Additionally, it presents a comparison of user task flows in SPHINX, password-only authentication, and other device-based password managers.

Future work could involve extending SPHINX to support an online instantiation, providing further security against device compromise and man-in-the-middle attacks. However, building and testing the full implementation of such an extension are beyond the current scope.

1.2 Problem Statement:

Existing password management systems are susceptible to compromise, leaving users vulnerable to various cyberattacks. Traditional solutions often generate low-entropy passwords and rely on plaintext entry of master passwords, making them targets for online guessing and offline dictionary attacks. Additionally, compromised password managers can lead to unauthorized access to sensitive user information, posing significant risks to individuals and organizations alike. With the ever-increasing sophistication of cyber threats and the widespread adoption of digital services, the need for robust password management solutions has become more critical than ever. Addressing these vulnerabilities is essential to safeguarding user credentials, maintaining privacy, and protecting against financial loss and reputational damage.

1.3 Problem Illustration:

Consider a scenario where a user, Alice, relies on a traditional password manager to store and generate passwords for her various online accounts. Despite her best efforts to create strong passwords, the password manager she uses has inherent vulnerabilities that leave her susceptible to cyberattacks.

One day, Alice receives a phishing email that tricks her into clicking on a malicious link, compromising her password manager's security. Unbeknownst to Alice, the attacker gains access to her master password and all the stored credentials for her online accounts. With full control over Alice's password manager, the attacker can now freely access and exploit her sensitive information.

Furthermore, the passwords generated by Alice's password manager have low entropy, making them vulnerable to brute-force attacks and dictionary-based hacking techniques. Even if Alice had used unique passwords for each account, the compromised password manager puts all her accounts at risk of unauthorized access and potential data breaches.

As a result of the security breach, Alice faces significant consequences. Her personal and financial information could be stolen, leading to identity theft, financial fraud, or unauthorized access to sensitive data. Moreover, Alice's trust in online security is shattered, impacting her confidence in digital services and her overall peace of mind.

This scenario highlights the pressing need for password management solutions that offer robust security measures to protect user credentials, even in the face of compromise. Without such solutions, individuals like Alice remain at risk of falling victim to cyberattacks and the devastating consequences that follow.

2. LITERATURE SURVEY

The landscape of password management has evolved significantly over the years, with SPHINX emerging as a novel application of Device-Enhanced Password-Authenticated Key Exchange (DE-PAKE) principles. This survey provides an overview of existing literature in the field, highlighting key research contributions and comparing them with the features and advantages offered by SPHINX.

Traditional Password Managers: Traditional password managers have been widely used, offering basic password storage and generation capabilities. However, they often rely on low-entropy passwords and plaintext entry of master passwords, leaving users vulnerable to various cyber threats, including phishing attacks, dictionary attacks, and compromise of the password manager itself.

DE-PAKE Principles: DE-PAKE, as introduced in [31], provides a foundation for SPHINX by securely transforming user-memorable passwords into high-entropy random strings. This cryptographic primitive addresses the limitations of traditional password management systems by leveraging a secondary device to generate strong, unique passwords for each service.

Security Features of SPHINX: SPHINX offers several key security guarantees simultaneously, including resistance to online guessing attacks, offline dictionary attacks under server and device compromise, phishing attacks, and eavesdropping or man-in-the-middle attacks on the device-client channel. These features enhance the overall security posture of password management systems and mitigate common vulnerabilities associated with traditional approaches.

Usability Advantages of SPHINX: In addition to its robust security features, SPHINX offers usability advantages such as the use of human-memorable passwords, easy password updates through device key management, and compatibility with multiple types of devices, including smartphones, wearables, and online services. These usability enhancements improve user experience and facilitate seamless integration into daily routines.

Comparative Analysis: Comparative evaluations of SPHINX with traditional password managers and other password management schemes highlight the unique advantages offered by SPHINX. While traditional solutions may provide some security and usability features, SPHINX stands out for its comprehensive security guarantees and enhanced usability, particularly in scenarios involving compromised servers or devices.

Future Directions: Future research in password management may explore further extensions and applications of DE-PAKE principles, as demonstrated by SPHINX. Additionally, ongoing efforts to improve password security and usability will continue to shape the landscape of password management solutions, with SPHINX serving as a compelling example of innovation in the field.

In summary, SPHINX represents a significant advancement in password management, offering a comprehensive solution that addresses both security and usability challenges inherent in traditional approaches. Its innovative application of DE-PAKE principles and robust security features position it as a promising solution for enhancing password security in an increasingly digital world.

Existing Method:

Cracking-resistant password encoding strategies have been proposed in the literature to render offline dictionary attacks ineffective. They introduce the notion of out putting decoy passwords to an attacker who compromises the manager and attempts to decrypt the passwords with a wrong master password. Since the attacker is not aware of the correct password, any attempt to login with the decoy passwords can be prevented on the server, and raise an alert.

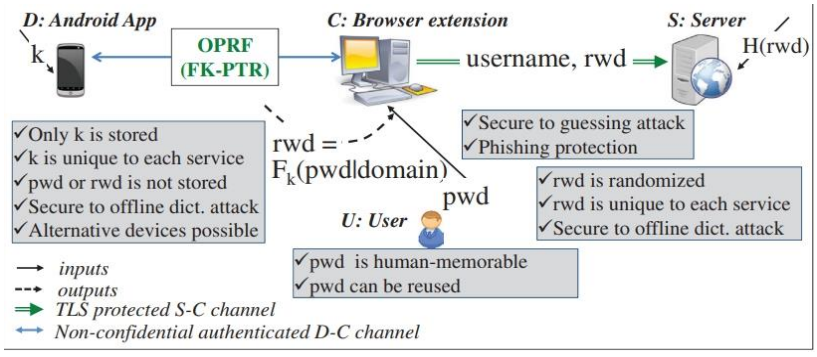


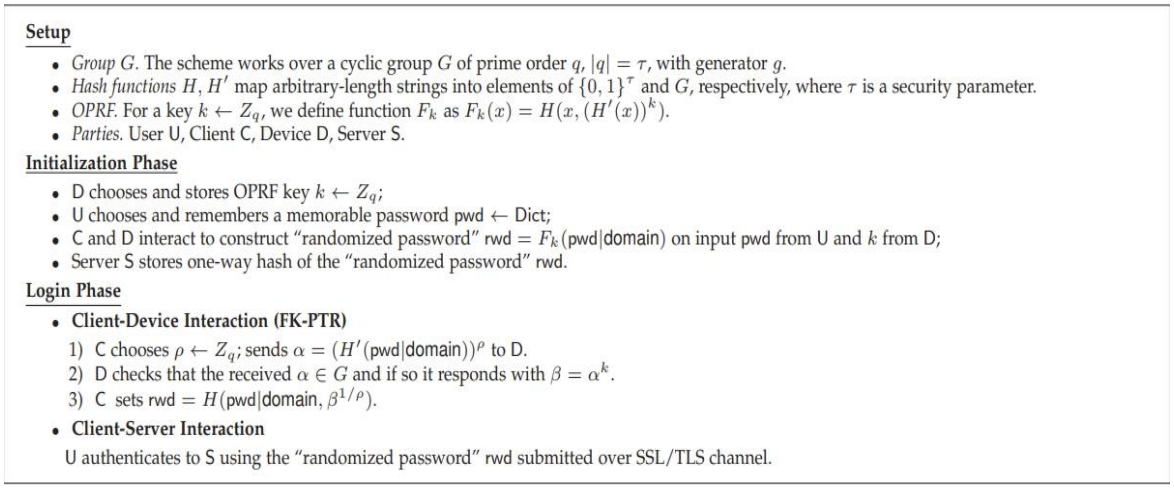
Figure 2.1. Existing method

SI.NO	Strategies	Advantages	Disadvantages
1	On the security of cracking-resistant password vaults.	investigate the construction of encrypted vaults that resist such offline cracking attacks and force attackers instead to mount online attacks	A password vault can greatly reduce the burden on a user of remembering passwords, but introduces a single point of failure
2	Round-optimal password-protected secret sharing and T-PAKE in the password only model	the system is secure against offline password attacks by an attacker controlling up to t servers	Time taking and require sever authentication
3	Device Enhanced Password Protocols with Optimal Online-Offline Protection.	An attacker taking over the device still requires a full online attack to impersonate the user.	Hardware device is required for using this method.
4	A Comparative Usability Evaluation of Traditional Password Managers	users were not comfortable giving control of their passwords to an online entity and preferred to manage their passwords themselves on their own portable device	Use apps to manage user passwords which is not secure
5	Cracking resistant password vaults using natural language encoders.	a new type of secure encoding scheme that we call a natural language encoder (NLE). An NLE permits the construction of vaults which, when decrypted with the wrong master password, produce plausible looking decoy passwords.	Use automated NLP technique which is defied by system
6	The emperor's new password manager: Security analysis of web-based password managers	We identify four key security concerns for web-based pass-word managers and, for each, identify representative vulnerabilities through our case studies	Data analysis on various attacks is performed

Table 2.1. Literature Survey

3. PROPOSED METHOD

3.1. Illustration



Setup

- *Group* G. The scheme works over a cyclic group G of prime order q, $|q| = \tau$, with generator g.
- *Hash functions* H, H' map arbitrary-length strings into elements of $\{0,1\}^\tau$ and G, respectively, where τ is a security parameter.
- *OPRF.* For a key $k \leftarrow Z_q$, we define function F_k as $F_k(x) = H(x, (H'(x))^k)$.
- *Parties.* User U, Client C, Device D, Server S.

Initialization Phase

- D chooses and stores OPRF key $k \leftarrow Z_q$;
- U chooses and remembers a memorable password pwd $\leftarrow$ Dict;
- C and D interact to construct "randomized password" rwd $= F_k(\text{pwd}|\text{domain})$ on input pwd from U and k from D;
- Server S stores one-way hash of the "randomized password" rwd.

Login Phase

- **Client-Device Interaction (FK-PTR)**
  1) C chooses $\rho \leftarrow Z_q$; sends $\alpha = (H'(\text{pwd}|\text{domain}))^\rho$ to D.
  2) D checks that the received $\alpha \in G$ and if so it responds with $\beta = \alpha^k$.
  3) C sets rwd $= H(\text{pwd}|\text{domain}, \beta^{1/\rho})$.
- **Client-Server Interaction**

  U authenticates to S using the "randomized password" rwd submitted over SSL/TLS channel.

Figure 3.1.1. SPHINX

3.2. Authentication and User Management:

3.2.1. User Registration:

User registration is the process by which individuals sign up for access to the password management system. During registration, users provide their email address and create a master password. This master password serves as the primary authentication credential for accessing the system and managing their passwords. Additionally, to verify the authenticity of the provided email address and enhance security, an OTP (one-time password) is sent to the user's email during the registration process. The user must enter this OTP to confirm their email address and complete the registration process successfully. This verification step helps ensure that only users with valid email addresses can register for the system, mitigating the risk of unauthorized access and account creation by malicious actors.

3.2.2. User Login:

User login enables registered users to access their accounts and utilize the functionalities of the password management system. To log in, users enter their registered email address and master password into the login interface. This master password is securely stored and used for authentication purposes. In addition to the master password, an extra layer of security is implemented through OTP verification. Upon entering their credentials, a one-time password is generated and sent to the user's registered email address. The user must then retrieve this OTP from their email and enter it into the login interface to authenticate their identity fully. This two-factor authentication mechanism enhances the security of the login process, reducing the risk of unauthorized access even if the user's password is compromised.

3.2.3. Admin Panel:

The admin panel provides administrators with the necessary tools and functionalities to manage user accounts and oversee the operation of the password management system. Within the admin panel, administrators have the capability to view and manage user accounts, including creating new accounts, modifying account details, and deactivating or deleting accounts as needed. Additionally, administrators can reset passwords for user accounts in cases where users forget their master passwords or require assistance with account access. Furthermore, the admin panel allows administrators to manage user roles, assigning specific permissions and privileges to different user accounts based on their roles within the organization or system. This role-based access control enhances security and ensures that users have appropriate levels of access to system resources and functionalities based on their responsibilities and requirements. Overall, the admin panel serves as a centralized hub for administrators to efficiently manage user accounts and maintain the security and integrity of the password management system.

3.3. Password Management:

Password Encryption:

Password encryption is a crucial aspect of the password management system, ensuring that user passwords are securely stored and protected from unauthorized access. The process involves several steps:

RSA Key Pair Generation:

Upon user registration, a unique RSA key pair is generated for each user. This consists of a public key and a private key.

OPRF Technique for Key Derivation:

The user's master password is used as input to the Oblivious Pseudo-Random Function (OPRF) technique.

OPRF derives a cryptographic key from the master password without revealing the password itself.

Encryption of Derived Key:

The derived cryptographic key is encrypted using the user's public RSA key.

This ensures that only the corresponding private RSA key, held by the user, can decrypt the key.

AES Encryption for Password Storage:

User passwords are encrypted using Advanced Encryption Standard (AES) encryption.

The encrypted passwords are stored in the database, ensuring that even if the database is compromised, passwords remain secure.

Password Retrieval:

When a user needs to retrieve a password, the stored encrypted key and password are decrypted using the following steps:

Decryption of Stored Key:

The encrypted key stored in the database is decrypted using the user's private RSA key. This retrieves the original derived cryptographic key.

Decryption of Encrypted Password:

The retrieved cryptographic key is used to decrypt the encrypted password stored in the database. The decrypted password is then made available to the user.

Password Generation:

The password management system also provides functionality for generating strong, random passwords. Users can customize various parameters for the generated passwords, including:

Length: Users can specify the desired length of the generated password, allowing for flexibility based on security requirements or platform constraints.

Character Set: Users can choose the character set from which the password will be generated, including options such as uppercase letters, lowercase letters, numbers, and special characters.

By offering these customization options, users can generate passwords tailored to their specific needs while ensuring they meet the desired level of security.

Additionally, the use of randomness in password generation helps mitigate the risk of predictability and enhances the overall strength of generated passwords, further bolstering the security of the password management system.

3.4. OTP Verification:

OTP Generation and Sending:

OTP (One-Time Password) generation and sending are integral parts of the two-factor authentication process implemented in the password management system. This process involves the following steps:

OTP Generation:

Upon user login or any action requiring OTP verification, a unique one-time password is generated by the system. The OTP is typically a randomly generated string of alphanumeric characters or digits, ensuring unpredictability and security.

Sending OTP to User's Registered Email:

Once generated, the OTP is sent to the user's registered email address.
The email serves as a secure channel for delivering the OTP to the user, leveraging existing communication infrastructure and ensuring that the OTP remains confidential.

User Login Attempt:

When users attempt to log in or perform a sensitive action requiring OTP verification, they are prompted to enter the OTP received via email.

Entering OTP:

Users retrieve the OTP from their email inbox and enter it into the designated field in the login interface. The OTP entered by the user is compared against the OTP generated and sent by the system.

Validation and Authentication:

If the entered OTP matches the one generated and sent by the system and it is within the valid time period, authentication is successful. Upon successful OTP verification, users are granted access to their accounts or allowed to perform the requested action. OTP verification adds an additional layer of security to the authentication process, requiring users to possess both their master password and access to their registered email account. This two-factor authentication mechanism enhances the overall security posture of the password management system, reducing the likelihood of unauthorized access by malicious actors even in the event of password compromise.

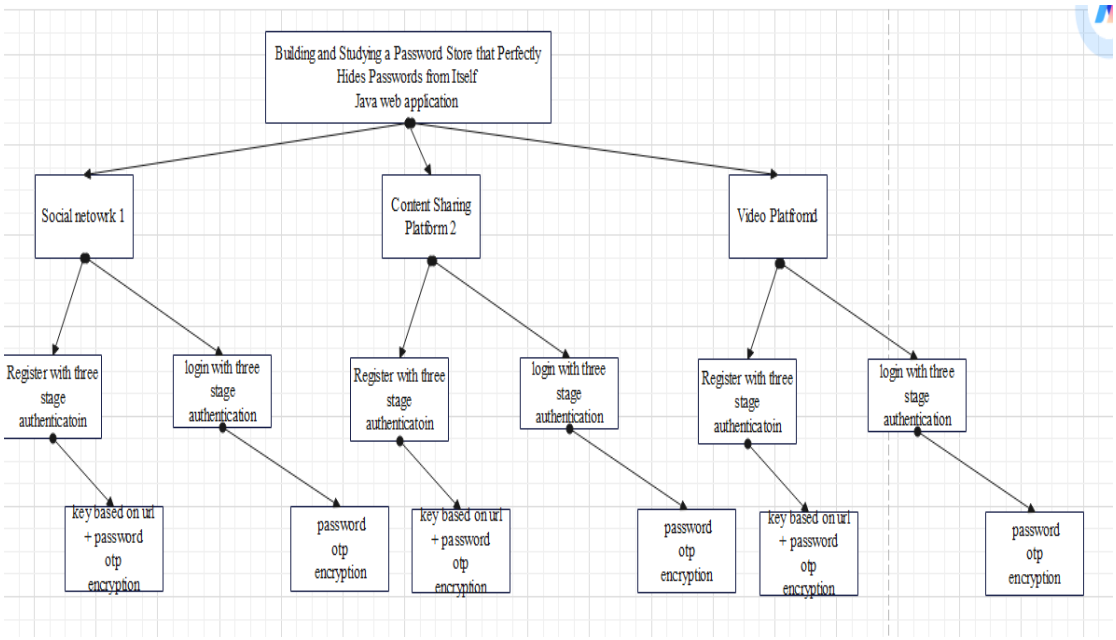


Fig 3.2.1. Authentication and User Management

3.5. USER INTERFACE:

Chrome Extension Interface:

The Chrome extension interface provides users with a seamless and intuitive experience directly within their web browser. This interface is designed to be user-friendly and straightforward, allowing users to easily access and manage their passwords. Key features of the Chrome extension interface include:

Simple and Intuitive Design:

The interface features a clean and minimalist design, ensuring that users can quickly navigate and understand its functionalities. Intuitive navigation elements, such as buttons and menus, make it easy for users to perform actions like saving, retrieving, and generating passwords.

Password Management Options:

Users can save passwords for various online accounts directly from the extension interface. Retrieval of saved passwords is also straightforward, with users able to quickly access their stored passwords whenever needed. Additionally, the extension provides functionality for generating strong and random passwords, offering users an easy way to create secure passwords for new accounts or password updates.

Web Interface:

The web interface complements the Chrome extension by providing administrators with access to the admin panel and user account management functionalities. This web interface is built using Django, offering a robust and feature-rich platform for managing the password management system. Key aspects of the web interface include:

Admin Panel Access:

Administrators can access the admin panel through the web interface to perform various administrative tasks, such as managing user accounts, resetting passwords, and assigning user roles. The admin panel provides administrators with a centralized dashboard for overseeing system operations and user activities.

Responsive Design:

The web interface is designed with responsiveness in mind, ensuring that it adapts seamlessly to different screen sizes and devices. Whether accessed from a desktop computer, tablet, or smartphone, the web interface provides a consistent and optimized user experience.

By combining the Chrome extension interface for users with the web interface for administrators, the password management system offers a comprehensive solution for securely managing passwords. Users benefit from a user-friendly interface within their web browser, while administrators have access to powerful management tools through the web interface, all built on the Django framework for stability and scalability.

3.6. SECURITY MEASURES

Data Encryption:

In our implementation, we prioritize the security of stored passwords by employing AES encryption. This encryption technique ensures that passwords stored in the database are protected from unauthorized access, even if the database is compromised. By encrypting passwords using AES, we add an extra layer of defence, making it extremely difficult for attackers to decipher sensitive information.

Additionally, we enforce secure communication between the Chrome extension and the server by using HTTPS protocol. This ensures that data transmitted between the client and server is encrypted, preventing eavesdropping and man-in-the-middle attacks. By adhering to HTTPS standards, we maintain the confidentiality and integrity of user data throughout the communication process.

Input Validation:

To mitigate common security vulnerabilities such as cross-site scripting (XSS) and SQL injection, we implement rigorous input validation mechanisms. Our system validates user input at various entry points, including registration forms, login interfaces, and password retrieval requests. By sanitizing and validating user input, we prevent malicious actors from injecting malicious scripts or SQL queries into our system. This helps safeguard against potential attacks aimed at exploiting vulnerabilities in our application, ensuring the integrity and security of user data.

Rate Limiting and Brute Force Protection:

We incorporate rate limiting measures to mitigate the risk of brute force attacks targeting login and OTP verification processes. By limiting the number of login attempts within a specific timeframe, we prevent malicious actors from repeatedly guessing user credentials or OTPs. This helps safeguard user accounts from unauthorized access attempts and strengthens overall system security.

Additionally, we implement brute force protection mechanisms to detect and respond to suspicious login patterns indicative of brute force attacks. By monitoring login attempts and identifying anomalous behaviour, our system can automatically block or throttle access for suspicious IP addresses, mitigating the impact of brute force attacks on system security. These proactive measures help fortify our system against brute force attacks and enhance the overall security posture of our password management solution.

4. DESIGN

4.1. UML Diagrams

UML, or Unified Modeling Language, is a graphical tool essential for designing software systems. It offers standardized visual models for representing object-oriented software structures. UML diagrams are crucial for clear and organized communication of design concepts. These diagrams are indispensable in software design, aiding developers in understanding and analyzing intricate systems. They serve as effective tools for conveying design ideas to team members, stakeholders, and clients, ensuring that the software meets required standards of functionality, performance, and quality. Moreover, UML diagrams help in detecting and rectifying errors early in the development process, thereby saving time and reducing costs. They come in two main types: structural diagrams, which depict the static structure of the system, and behavioral diagrams, which illustrate dynamic interactions and Workflows. In summary, UML diagrams play a vital role in streamlining the software development process, providing developers with a clear and efficient means of conceptualizing and refining software systems.

4.1.1. Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

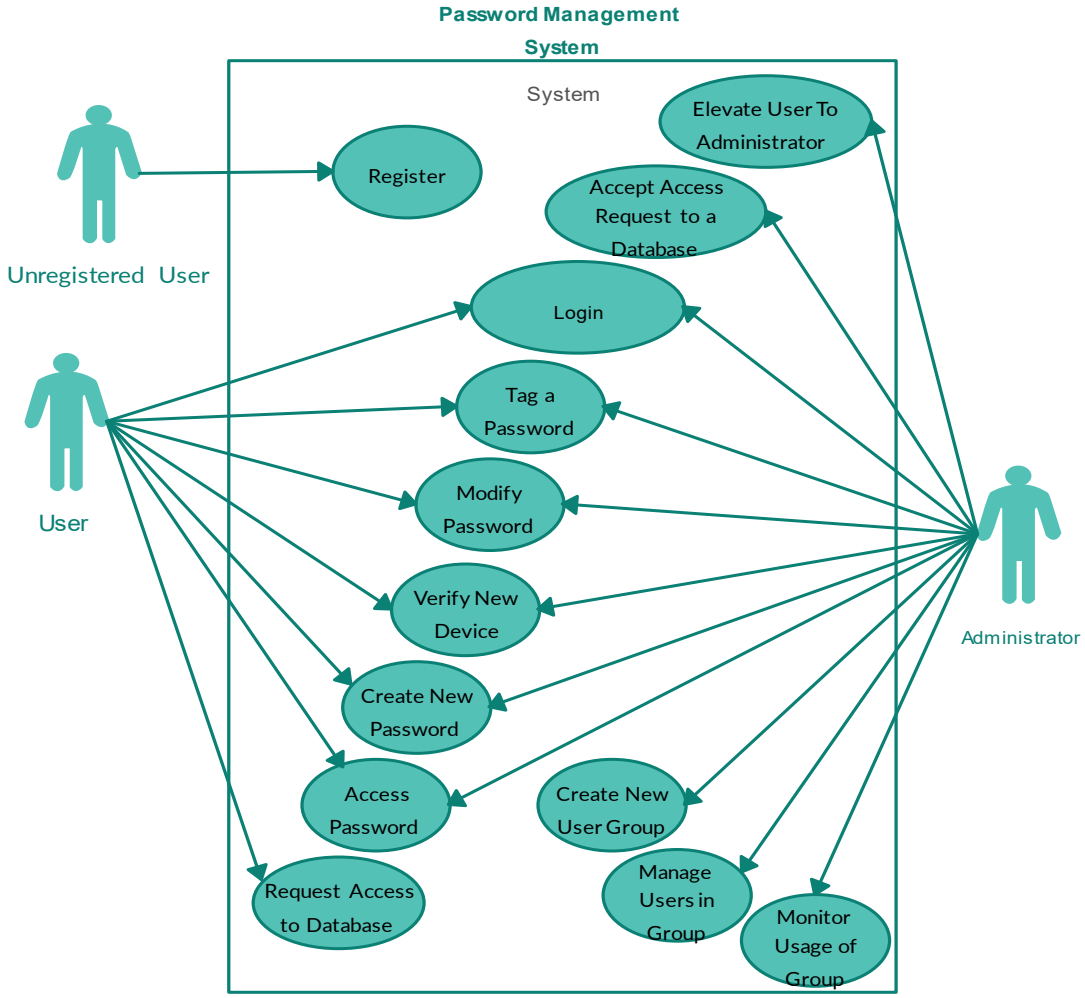


Figure 4.1.1.1 Use case Diagram

1. **User:**
    - Description: Represents individuals who utilize the Sphinx password management system to securely store and manage their passwords.
    - Responsibilities:
        - Register for a new account, providing necessary information such as email and creating a master password.
        - Log in securely using their credentials.
        - Access and manage stored passwords, utilizing functionalities such as adding, retrieving, and updating passwords.
        - Utilize the DE-PAKE model for secure separation of master passwords from stored information, ensuring confidentiality even on compromised devices.
        - Benefit from the enhanced security provided by AES and RSA encryption for password storage and communication.
        - Experience high-entropy password transformation through the OPRF scheme, enhancing resistance against online attacks.
2. **Admin:**
    - Description: Represents administrators or system managers responsible for overseeing the Sphinx password management system.
    - Responsibilities:
        - Log in securely using admin credentials to access administrative functionalities.
        - Manage user accounts, including registration, modification, and deactivation.
        - Monitor system activities and perform necessary administrative tasks to ensure smooth operation.
        - Utilize the DE-PAKE model to ensure separation of user master passwords from stored information, maintaining security and confidentiality.

- Ensure adherence to security protocols and best practices, including AES and RSA encryption, for enhanced system security.
- Oversee the implementation of the OPRF scheme for high-entropy password transformation, contributing to robust security measures.

3. **Attacker:**
    - Description: Represents malicious entities attempting to exploit vulnerabilities in the Sphinx password management system for unauthorized access or data breaches.
    - Responsibilities:
        - Attempt to gain unauthorized access to user accounts or system resources using various techniques, including brute force attacks or exploitation of vulnerabilities.
        - Exploit weaknesses in the system's security measures, such as AES and RSA encryption, to access sensitive information or compromise user accounts.
        - Target the DE-PAKE model to circumvent security measures and gain access to stored passwords or user data.
        - Attempt to disrupt system operations or compromise user privacy through malicious activities.
        - Encounter resistance from the OPRF scheme, which enhances password security and makes it more challenging to compromise user accounts through online attacks.

4.1.2. Sequence Diagram

A sequence diagram is a visual representation that illustrates the interactions and communication flow between different components or objects in a system over time. It demonstrates how these entities collaborate and exchange messages to achieve specific functionalities. In a sequence diagram, the vertical axis typically represents time, while horizontal lines, called lifelines, represent individual entities or objects involved in the interaction. Arrows between lifelines indicate the flow of messages or method calls between these entities, depicting the sequence of actions and responses. Sequence diagrams are invaluable tools in software engineering for designing and understanding the behavior of systems, as they provide a clear and intuitive visualization of the runtime interactions between various components. They aid in identifying potential bottlenecks, understanding the order of operations, and ensuring that system requirements are met. Additionally, sequence diagrams facilitate communication among stakeholders by offering a concise and structured depiction of system behavior, enabling effective collaboration and decision-making during the software development process. Overall, sequence diagrams play a crucial role in analyzing, designing, and documenting the dynamic behavior of complex systems

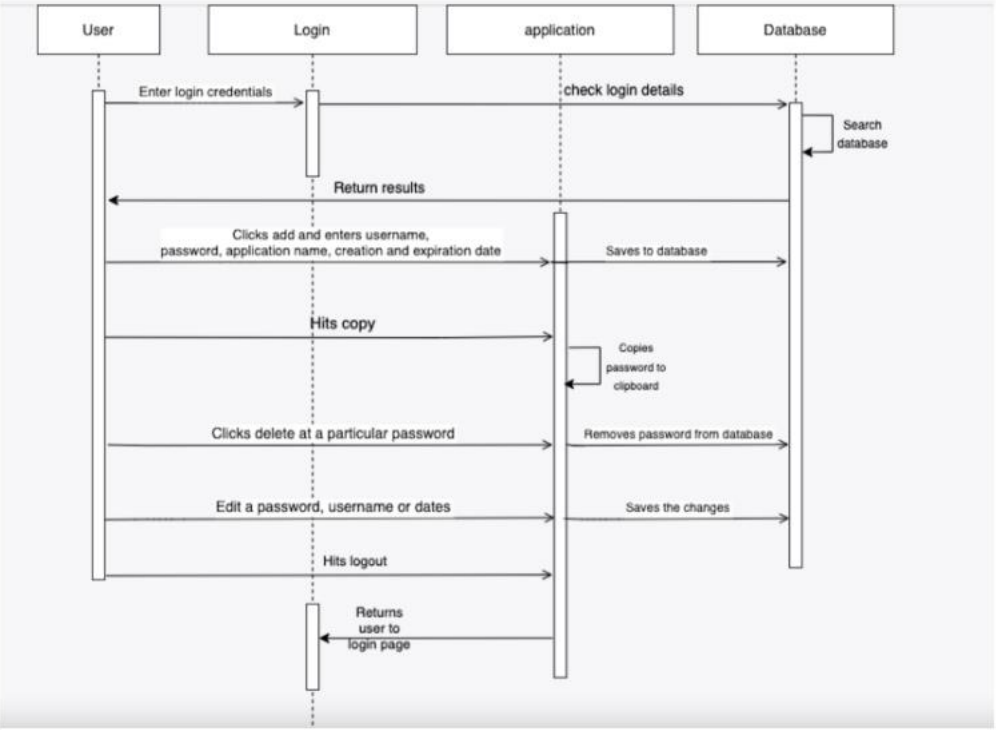


Figure 4.1.2.1. Sequence Diagram

1. **User Registration:**

- The sequence begins when the user initiates the registration process by providing their email and creating a master password.
- The system generates an OTP (One-Time Password) for email verification and sends it to the user's registered email address.
- The user receives the OTP and enters it into the registration interface for verification.
- Upon successful OTP verification, the user's registration is completed, and their account is created in the system.

2. **User Login:**

- The sequence starts when the user attempts to log in by providing their email and master password.
- The system verifies the provided credentials and generates an OTP for two-factor authentication.
- The OTP is sent to the user's registered email address.

- The user retrieves the OTP from their email and enters it into the login interface.

3. **Password Management:**

- Once logged in, the user can perform various password management tasks.
- To add a new password, the user provides the necessary details such as the website/application name and the password itself.
- The system encrypts the password using AES encryption before storing it securely.
- To retrieve a password, the user selects the desired website/application from their stored list.
- The system decrypts the stored password using AES encryption and provides it to the user.
- If the user wishes to update a password, they provide the new password, which is encrypted and stored in place of the old one.

4. **Admin Functionality:**

- An admin initiates the sequence by logging in with their admin credentials.
- Once logged in, the admin can access administrative functionalities such as managing user accounts.
- The admin can create new user accounts, modify existing account details, or deactivate accounts as needed.
- The admin has access to monitoring tools to oversee system activities and ensure smooth operation.

5. **Security Measures:**

- Throughout the sequence, the system implements various security measures to protect user data and ensure system integrity.
- AES encryption is used to securely store passwords and communicate sensitive information.
- RSA encryption is employed for secure communication between users and the system.
- The DE-PAKE model ensures the separation of user master passwords from stored information, preventing leakage even in compromised devices.
- The OPRF scheme enhances password security through high-entropy password transformation, making it resistant to online attacks.

4.1.3. Class Diagram

A class diagram serves as a visual blueprint for software systems, outlining the structure and connections between different components. In simple terms, it depicts the building blocks of a program, known as classes, and how they interact with each other. Each class represents a specific entity or object in the system, detailing its attributes (characteristics) and methods (actions). Associations between classes indicate relationships, illustrating how they collaborate to achieve system functionality. These diagrams are crucial for software development, providing a clear overview of the system's architecture, aiding in communication among developers, and facilitating the design and maintenance of complex software projects

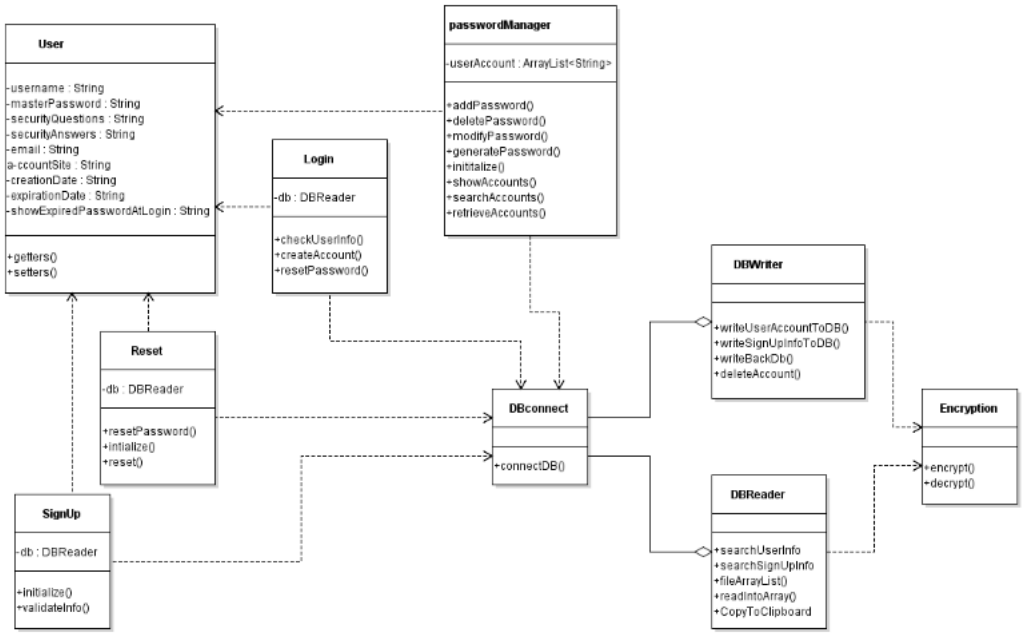


Figure 4.1.3.1 Class Diagram

4.1.4. Collaboration Diagram:

A collaboration diagram, also known as a communication diagram, is a type of interaction diagram in the Unified Modeling Language (UML) that illustrates the interactions and relationships between objects or components within a system to achieve a specific functionality or behavior. In a collaboration diagram, the emphasis is placed on how objects or components interact with each other to accomplish a task or respond to an event. It typically represents these interactions through messages exchanged between objects, which are depicted as labeled arrows between the objects. These messages can represent method calls, signals, or other forms of communication. The main purpose of a collaboration diagram is to visualize the dynamic aspects of a system's behavior, showing how objects collaborate to fulfill certain requirements or achieve specific objectives. By depicting the flow of messages between objects, developers can gain insights into the runtime behavior of the system and identify

potential design flaws or areas for optimization. Overall, collaboration diagrams serve as valuable tools for understanding and communicating the runtime behavior and interactions within a system, aiding in system design, analysis, and debugging processes.

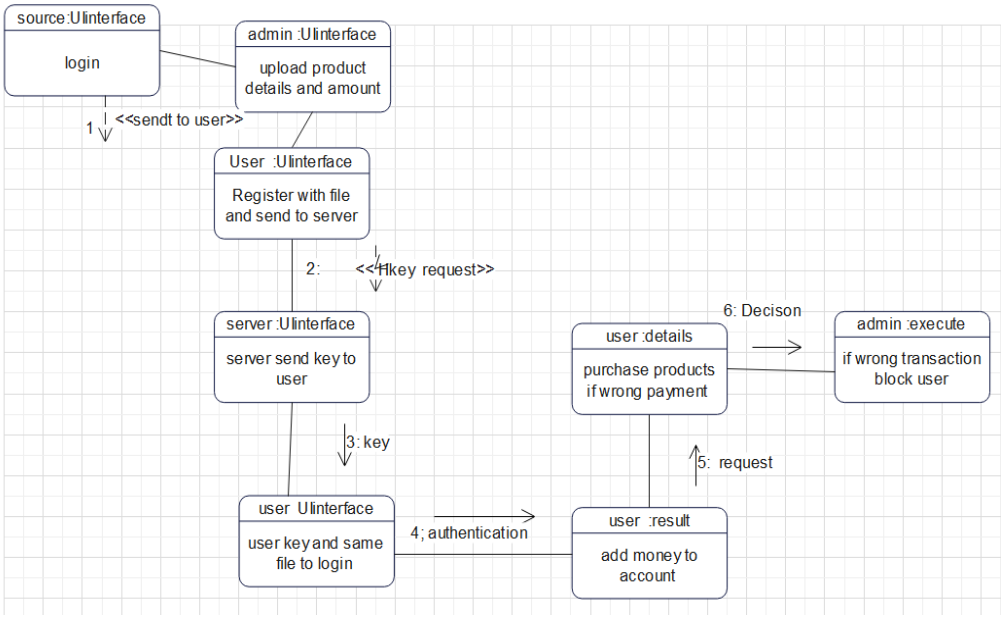


Figure 4.1.4.1 Collaboration Diagram

4.1.5. State Diagram:

A state diagram, also known as a state machine diagram or statechart diagram, is a graphical representation used to depict the various states of an object or system and the transitions between those states. It is a behavioral diagram that shows the different states an object can be in over time, as well as the events that cause transitions between these states. In a state diagram, each state is represented as a node, typically drawn as a rounded rectangle, and transitions between states are represented by arrows, showing the flow from one state to another. Events trigger these transitions, and conditions or actions associated with transitions may also be depicted. Additionally, the diagram may include initial and final states to denote the starting and ending points of the system's behavior.

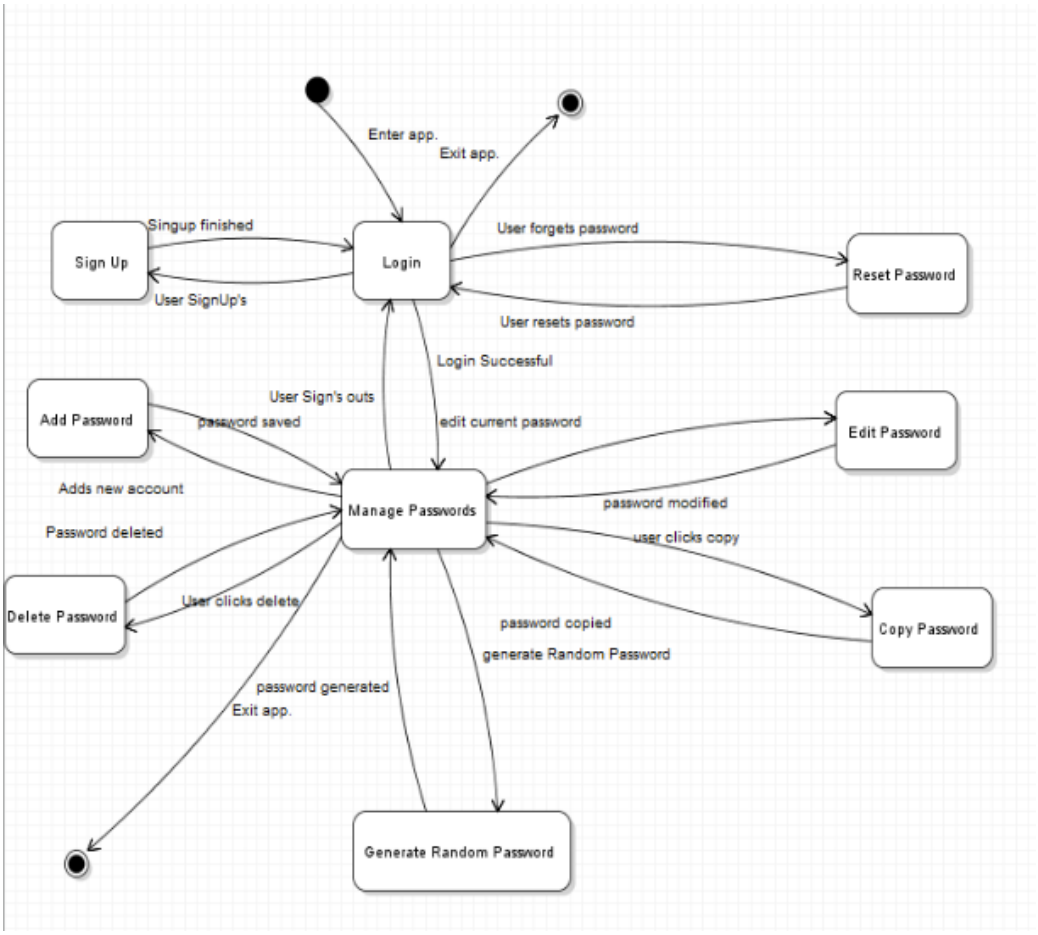


Figure 4.1.5.1. State Diagram

4.1.6. Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram used to visually represent the flow of control or the sequence of activities in a system, process, or workflow. It typically consists of various activities, represented by nodes, connected by arrows to indicate the sequence in which the activities are performed. Each activity can represent a single operation, a group of operations, or a decision point within the system. The arrows between activities show the order in which the activities are executed, with forks and joins indicating parallel or concurrent execution paths. Activity diagrams are useful for modeling complex business processes, software workflows, or any sequence of actions that involve multiple actors or components. They provide a clear and concise visualization of how the system functions, helping stakeholders to understand, analyze, and communicate the behavior and logic of the system effectively

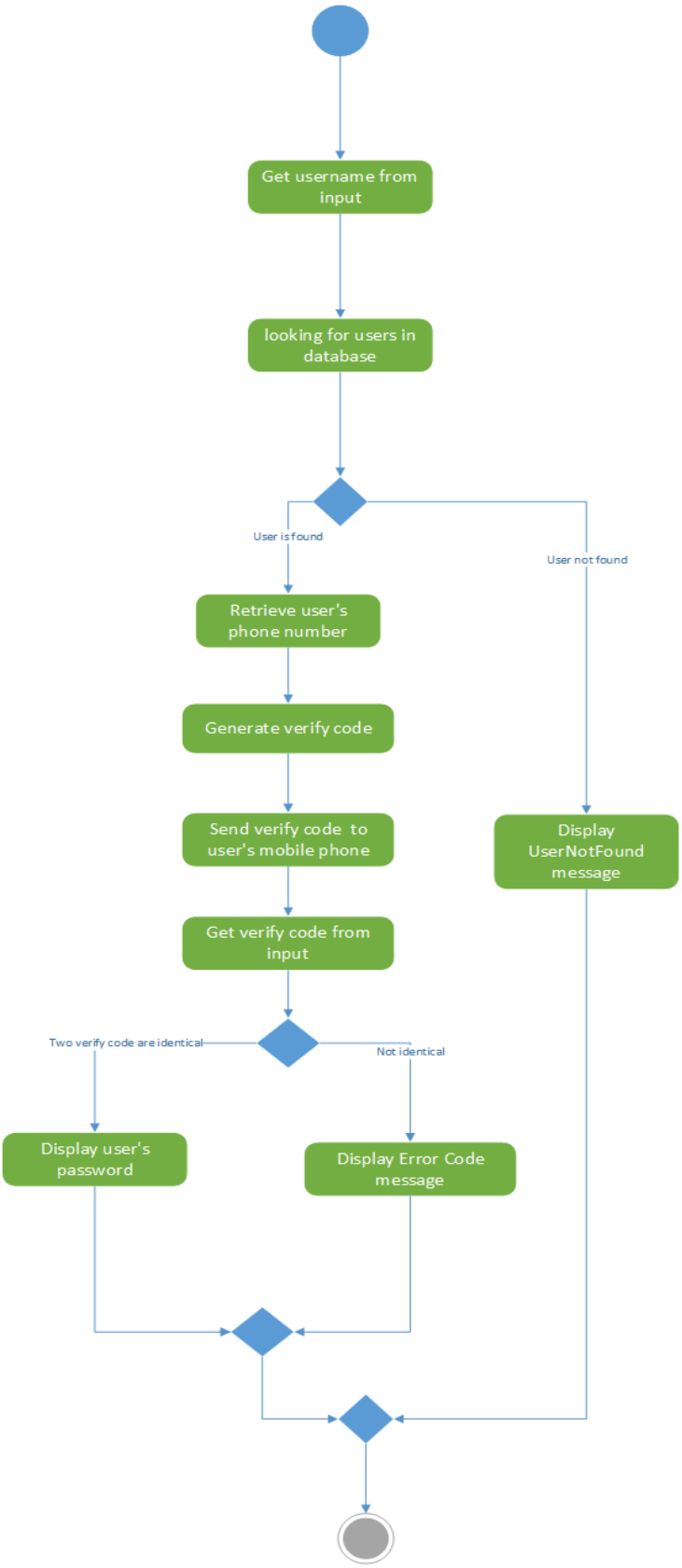


Figure 4.1.6.1 Activity Diagram

4.1.7. Component Diagram:

A component diagram is a type of UML (Unified Modeling Language) diagram that depicts the components of a system and their relationships. It illustrates how various software components or modules interact with each other to form a coherent system architecture. Components are represented as rectangles, each encapsulating the functionality they provide. Dependencies between components are depicted using arrows, indicating the direction of communication or dependency. Typically, a component diagram is used during the design phase of software development to visualize the high-level structure of the system and its constituent parts. It helps software architects and developers to understand the organization of the system, identify dependencies between components, and plan for integration and deployment. Additionally, component diagrams facilitate communication among stakeholders by providing a clear representation of the system's architecture and its components.

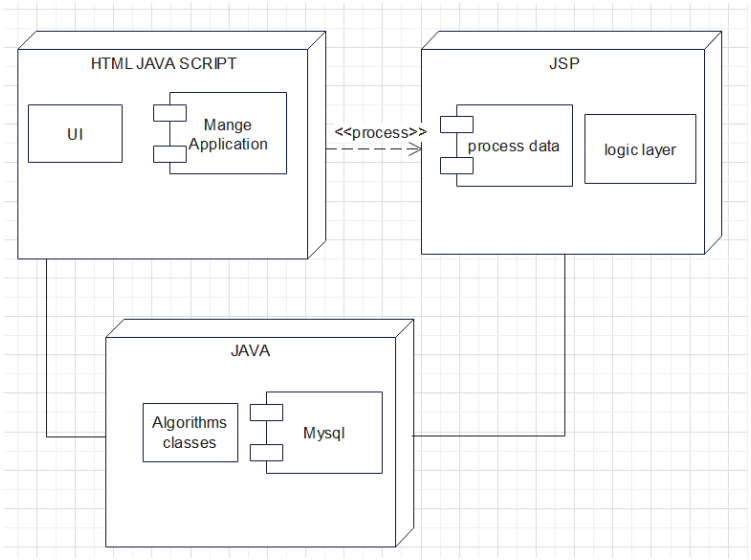


Figure 4.1.7.1 Component Diagram

4.1.8. Deployment Diagram

A deployment diagram is a type of UML (Unified Modeling Language) diagram that illustrates the physical deployment of software components within a computing environment. It shows how software artifacts, such as executable files, libraries, and databases, are distributed across hardware nodes, such as servers, workstations, or devices. Nodes are represented as boxes, while artifacts are depicted as rectangles, often connected by deployment relationships indicating which artifacts are hosted or executed on which nodes. Deployment diagrams are particularly useful for understanding the configuration and arrangement of software and hardware elements in a distributed system, including networks, servers, routers, and other infrastructure components. They help system architects and developers to plan for deployment, scalability, and performance optimization by visualizing how software components are distributed across different nodes and how they interact with each other over a network. Deployment diagrams also facilitate communication among stakeholders by providing a clear representation of the system's physical architecture and deployment topology.

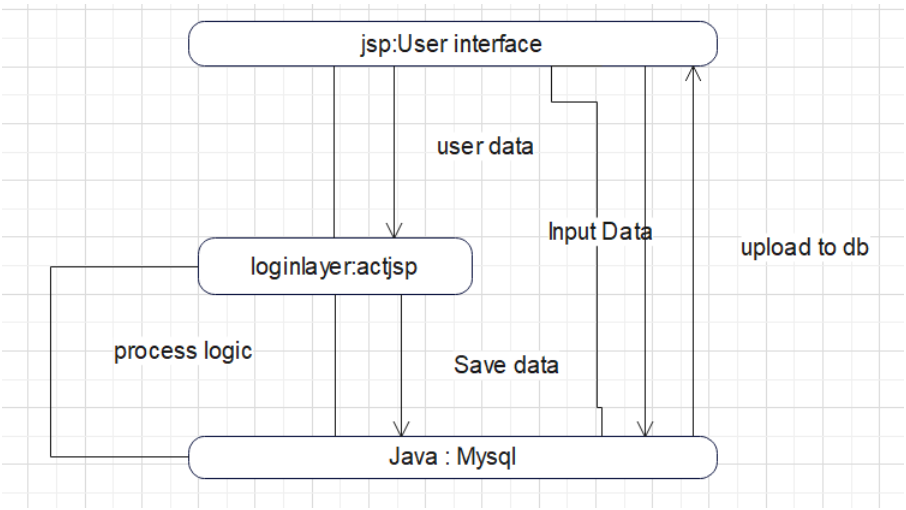


Figure 4.1.8.1 Deployment Diagram

5. IMPLEMENTATION

5.1. MODULES:

5.1.1. Overall Implementation:

Frontend Module:

In the frontend module of our project, we focus on developing the user interface components for the Chrome extension. This involves leveraging web technologies such as HTML, CSS, and JavaScript to create an intuitive and user-friendly interface. We design and implement various functionalities essential for password management directly within the extension interface. This includes features like user registration, login, password management, OTP verification, and password generation. Through carefully crafted UI elements and interactive elements, we aim to provide users with a seamless experience while interacting with the password management system. Additionally, we ensure that the frontend components are responsive and compatible across different devices and screen sizes, optimizing accessibility and usability for all users.

Backend Module:

In the backend module of our project, we establish the foundation for handling the core functionalities and business logic of the password management system. We set up a Django project, a powerful web framework for Python, to facilitate backend development. Within this Django project, we implement RESTful APIs using Django REST Framework, enabling seamless communication between the frontend and backend components of the system. These APIs serve as the bridge for transmitting data and requests between the client-side extension and the server-side application. Additionally, we focus on implementing robust business logic, data validation mechanisms, and database interactions within the backend. This includes tasks such as user authentication, password encryption, OTP verification, and user management. By meticulously designing and implementing the backend functionalities, we ensure the security, reliability, and scalability of our password management system, laying a solid foundation for its overall operation.

5.1.2. User Module:

User Authentication:

Within our system, user authentication serves as the first line of defense against unauthorized access. We implement a robust authentication mechanism utilizing both the master password and OTP verification, ensuring a multi-layered approach to user verification. This involves developing endpoints dedicated to user registration, login, and logout functionalities, enabling users to securely access their accounts. By incorporating OTP verification alongside the master password, we enhance the security posture of our system, mitigating the risk of unauthorized access. Through meticulous development and testing, we strive to deliver a seamless authentication experience, instilling confidence in users regarding the protection of their accounts and sensitive information.

Password Management:

Efficient and secure password management lies at the heart of our system's functionality. We implement a comprehensive suite of functionalities geared towards facilitating the seamless management of passwords for individual users. This includes features for saving, retrieving, and generating passwords, empowering users to effortlessly manage their credentials. To ensure the security of stored passwords, we implement robust encryption techniques and adhere to best practices for secure storage in the database. By prioritizing security without compromising usability, we aim to provide users with a reliable and intuitive platform for managing their passwords effectively.

User Interface Enhancements:

User interface enhancements play a pivotal role in shaping the overall user experience of our password management system. We prioritize the design of user-friendly interfaces that facilitate seamless interaction with password management features. Through thoughtful interface design, we aim to streamline user workflows and enhance usability. Additionally, we incorporate features such as password strength indicators and customization options for password generation, empowering users with tools to create and manage strong, unique passwords effortlessly. By continuously refining and optimizing the user interface, we strive to deliver a cohesive and intuitive user experience that fosters user engagement and satisfaction.

5.1.3. Admin Module:

Admin Module:

Within our system, the admin module is dedicated to facilitating administrative tasks and ensuring the smooth operation of the password management system. At its core, the admin module focuses on implementing role-based access control (RBAC) to distinguish between regular users and administrators, granting specific privileges to the latter. This includes defining admin privileges for managing user accounts, resetting passwords, and performing other administrative tasks essential for system management and security.

Admin Privileges:

One of the primary objectives of the admin module is to enforce role-based access control, delineating distinct roles and privileges for administrators. Through RBAC, we establish granular control over administrative actions, allowing administrators to perform tasks like managing user accounts, resetting passwords, and overseeing system settings. By defining specific admin privileges, we ensure that administrative actions are performed by authorized personnel, mitigating the risk of unauthorized access or misuse of administrative functionalities.

Admin Panel:

To facilitate administrative tasks, we develop a dedicated admin panel interface accessible only to authorized administrators. This admin panel serves as a centralized hub for administrators to perform various administrative tasks efficiently. We meticulously design interfaces within the admin panel for managing user accounts, viewing activity logs, and accessing system settings. By providing administrators with intuitive and user-friendly interfaces, we empower them to navigate and utilize administrative functionalities seamlessly, enhancing productivity and efficiency in managing the password management system.

Security Enhancements:

In addition to facilitating administrative tasks, the admin module incorporates robust security enhancements to safeguard sensitive administrative functionalities. This includes implementing additional security measures such as multi-factor authentication (MFA) and access control lists (ACLs) for admin functionalities. MFA adds an extra layer of security by requiring administrators to authenticate their identity through multiple factors, reducing the risk of unauthorized access. Furthermore, ACLs enable fine-grained control over access to administrative functionalities, ensuring that only authorized personnel can perform sensitive actions. By prioritizing security in admin functionalities, we reinforce the overall security posture of the password management system, safeguarding against potential security threats and unauthorized access.

5.1.4. Deployment:

As part of the deployment phase, our implementation plan entails deploying the Chrome extension to the Chrome Web Store for public distribution, ensuring widespread accessibility for users. Simultaneously, we deploy the backend server to a reliable hosting platform, prioritizing scalability and availability to accommodate varying user loads and maintain uninterrupted service. This involves configuring the server environment, database, and network settings meticulously for production deployment, guaranteeing optimal performance and security. By organizing the implementation plan into modules, we streamline development efforts, enabling focused attention on specific functionalities and components of the password management system. This structured approach fosters collaboration among team members, ensuring comprehensive coverage of all project aspects and ultimately delivering a robust and user-friendly password management solution to our users.

1. **AES Encryption for Password Storage:**

- After hashing the password into an elliptic curve using the browser extension's hashing mechanism, the resulting password (P) is encrypted using AES encryption:
- $Pencrypted = AESencrypt(P, keyAES)$
- Where:

Pencrypted is the encrypted password.

- *P* is the hashed password.
- *keyAES* is the AES encryption key.

2. **RSA Authentication for Secure Communication:**
    - To authenticate users and ensure secure communication, RSA encryption is used for encrypting user credentials (username and password) before transmission:
    - $C = RSAencrypt(Credentials, public\_keyRSA)$
    - Where:
        - C is the encrypted credentials.
        - $Credentials$ is the concatenated user credentials.
        - $public\_keyRSA$ is the public key of the system.
3. **Browser Extension for Hashing Password:**
    - The browser extension hashes the input password (P) into an elliptic curve using a "Hash-into-Elliptic-Curve" function: $H(P) = HEC(P)$
    - Where:
        - $H(P)$ is the hashed password.
        - HEC is the hash function for mapping the password onto an elliptic curve.
4. **Integration of Technologies:**
    - The integration of AES encryption, RSA authentication, and the browser extension's hashing mechanism ensures robust security measures for password management:
        - AES encryption protects stored passwords from unauthorized access:
        - $Pencrypted$
        - RSA encryption ensures secure communication by encrypting user credentials: C
        - The browser extension enhances password security during input by hashing passwords onto an elliptic curve: $H(P)$

5.2 Sample Code:

5.2.1 Home.html

```
!DOCTYPE html>!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Django Password Manager</title>
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">
   <link rel="stylesheet" href="{% static '/css/style.css' %}">
   <script src="{% static '/js/main.js' %}" defer></script>
</head>
<body>

<header>
   <nav>
      <div class="brand"><a href=""><i class="fa fa-lock" aria-hidden="true"></i> Password Manager</a></div>

      <ul>
         <li><a href="">Home</a></li>
         {% if request.user.is_anonymous %}
            <li><a href="javascript:;" onclick="displayModal('login-modal');">Login</a></li>
            <li><a href="javascript:;" onclick="displayModal('signup-modal');">SignUp</a></li>
```

```
        {% else %}
            <li><a href="javascript:;" onclick="document.getElementById('logout-
btn').click();">Logout</a></li>
            <form hidden="true" action="." method="POST">
              {% csrf_token %}
              <input type="submit" id="logout-btn" class="keyboard-input"
name="logout">
            </form>
            <li><a href="javascript:;"
onclick="displayPasswordModal();">View</a></li>
            <li><a href="javascript:;" onclick="displayModal('add-password-
modal');">Add</a></li>
        {% endif %}
      </ul>
   </nav>
</header>



   <!--Display messages-->
   {% if messages %}
    <div class="messages" style="text-align: center;">
      {% for message in messages %}
       <p>
          {% if message.tags == "error" %}
            <i class="fa fa-exclamation" aria-hidden="true"></i>
            {% else %}
            <i class="fa fa-check" aria-hidden="true"></i>
          {% endif %}
           {{message}}
```

```
            </p>
          {% endfor %}
      </div>
    {% endif %}

    <!--Confirm email-->
      {% if code %}
        <div style="text-align: center;">
            <form action="." method="POST" role="form">
                {% csrf_token %}
                <h2>Please confirm your email.</h2>
                <p>Check your email and get the code.</p>
                <input type="text" name="code" placeholder="enter the 6 digit code ...">
                <input type="text" hidden="true" name="user" value="{{user.username}}">
                <input type="submit" value="Confirm" name="confirm">
            </form>
        </div>
      {% endif %}




{% if loha %}
<div class="welcome-box" style="text-align:center">
    <h1>Welcome, {{ user.username }}!</h1>

    {% if url %}
        <p>URL: {{ url }}</p>
    {% endif %}
```

```
    {% if mode %}
        <p>Mode: {{ mode }}</p>
    {% endif %}

    {% if email %}
        <p>Email: {{ email }}</p>
    {% endif %}

    {% if password %}
        <p>Password: {{ password }}</p>
    {% endif %}
</div>
{% endif %}
    {% if activator %}
    <div style="text-align: center;">
        <form action="." method="POST" role="form">
            {% csrf_token %}
            <h2>Please confirm the code to login into {{ url }}.</h2>
            <p>Check your email and get me the code.</p>
            <input type="text" name="mode" placeholder="enter the 6 digit code ...">
            <input type="text" name="passw" placeholder="enter the encrypted pasword
...">

            <input type="text" hidden="true" name="user" value="{{user.username}}">
            <input type="submit" value="" name="organo">
        </form>
    </div>
    {% endif %}

    <!-- Your existing HTML code -->
```

```
    <!--Modals-->
    <div class="modals-wrapper">
        <span id="close-modal" title="close"><i class="fa fa-times" aria-hidden="true"></i></span>
        <!--login modal-->
        <div class="modal" id="login-modal">
            <form action="." role="form" method="POST">
                {% csrf_token %}
                <h2>Login</h2>
                <input type="text" name="username" placeholder="username">
                <input type="password" name="password" placeholder="password">
                <input type="submit" value="login" name="login-form">
            </form>
        </div>

        <!--signup modal-->
        <div class="modal" id="signup-modal">
            <form action="." role="form" method="POST">
                {% csrf_token %}
                <h2>SignUp</h2>
                <input type="text" name="username" placeholder="username">
                <input type="email" name="email" placeholder="email">

                <input type="password" name="password" placeholder="password">
                <input type="password" name="password2" placeholder="confirm password">
```

5.2.2 manage.py

```
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'pwdmngr.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

5.2.3 Views.py

```
from cryptography.fernet import InvalidToken
from django.contrib.auth.models import User
from django.shortcuts import render
from django.conf import settings
from django.contrib import messages
from django.http import HttpResponseRedirect
from django.contrib.auth import authenticate, login, logout
import random
from django.core.mail import send_mail
from cryptography.fernet import Fernet
from mechanize import Browser
import favicon
from .models import Password
import secrets
import string
import base64
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding

def encrypt(key, plaintext):
    backend = default_backend()
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
    encryptor = cipher.encryptor()

    # Ensure the plaintext is padded to a multiple of the block size
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_plaintext = padder.update(plaintext) + padder.finalize()
```

```
    ciphertext = encryptor.update(padded_plaintext) + encryptor.finalize()
    return ciphertext

def decrypt(key, ciphertext):
    backend = default_backend()
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
    decryptor = cipher.decryptor()

    padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()

    # Unpad the plaintext to get the original message
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    plaintext = unpadder.update(padded_plaintext) + unpadder.finalize()

    return plaintext

def generate_random_key(length):
    characters = string.ascii_letters + string.digits
    random_key = ''.join(secrets.choice(characters) for _ in range(length))
    # Ensure the key size is at least 16 bytes (128 bits)
    while len(random_key.encode('utf-8')) < 16:
        random_key += secrets.choice(characters)
    return random_key.encode('utf-8')

br = Browser()
br.set_handle_robots(False)

def myView(request):
```

```
if request.method == "POST":
    if "signup-form" in request.POST:
        username = request.POST.get("username")
        email = request.POST.get("email")
        password = request.POST.get("password")
        password2 = request.POST.get("password2")
        #if password are not identical
        if password != password2:
            msg = "Please make sure you're using the same password!"
            messages.error(request, msg)
            return HttpResponseRedirect(request.path)
        #if username exists
        elif User.objects.filter(username=username).exists():
            msg = f"{username} already exists!"
            messages.error(request, msg)
            return HttpResponseRedirect(request.path)
        #if email exists
        elif User.objects.filter(email=email).exists():
            msg = f"{email} already exists!"
            messages.error(request, msg)
            return HttpResponseRedirect(request.path)
        else:
            User.objects.create_user(username, email, password)
           new_user = authenticate(request, username=username, password=password2)
            if new_user is not None:
                login(request, new_user)
                msg = f"{username}. Thanks for subscribing."
                messages.success(request, msg)
                return HttpResponseRedirect(request.path)
    elif "logout" in request.POST:
```

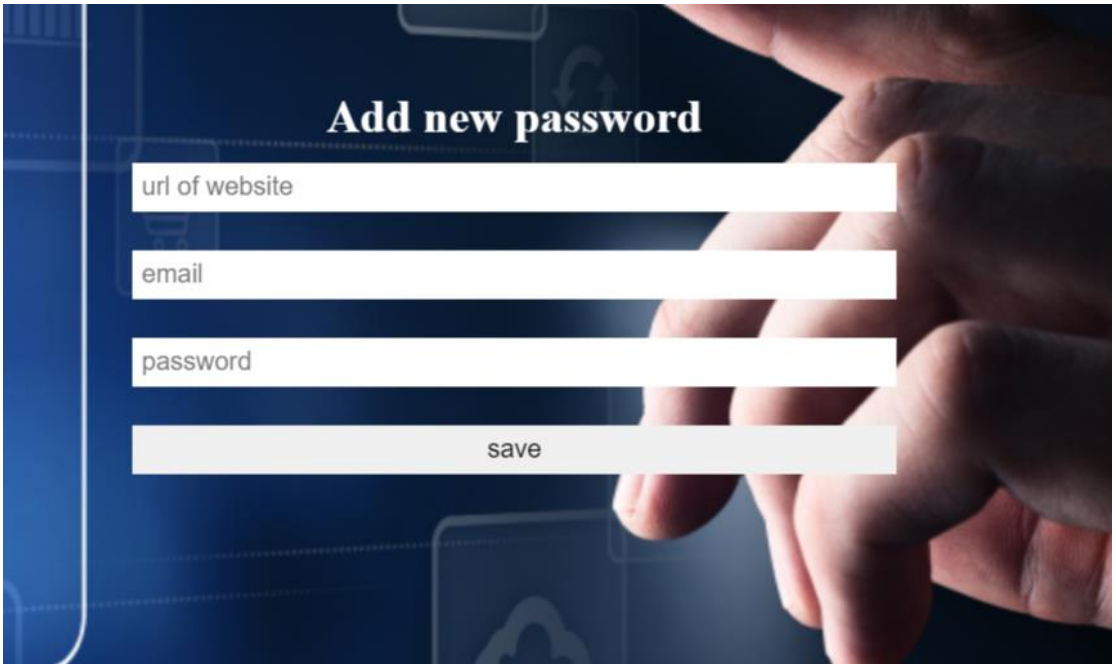
```
        msg = f"{request.user}. You logged out."
        logout(request)
        messages.success(request, msg)
        return HttpResponseRedirect(request.path)

    elif 'login-form' in request.POST:
        username = request.POST.get("username")
        password = request.POST.get("password")
        new_login = authenticate(request, username=username, password=password)
        if new_login is None:
            msg = f"Login failed! Make sure you're using the right account."
            messages.error(request, msg)
            return HttpResponseRedirect(request.path)
        else:
            code = str(random.randint(100000, 999999))
            global global_code
            global_code = code
            send_mail(
                "Django Password Manager: confirm email",
                f"Your verification code is {code}.",
                settings.EMAIL_HOST_USER,
                [new_login.email],
                fail_silently=False,
            )
            return render(request, "home.html", {
                "code":code,
                "user":new_login,
            })
```

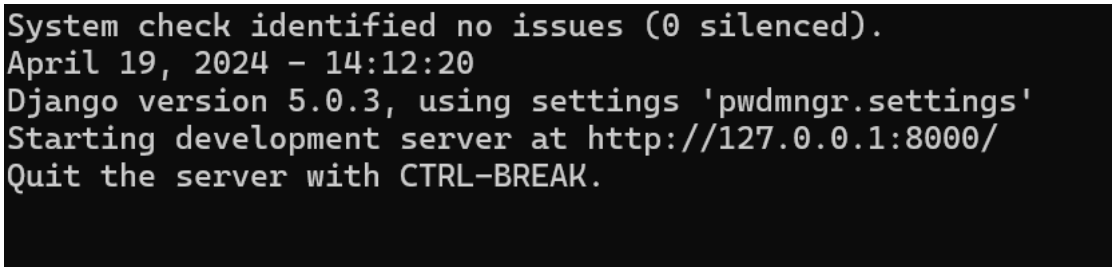
6. EXPERIMENT SCREENSHOTS

6.1 Home Page

6.2 Adding Passwords:



6.3 Server:


System check identified no issues (0 silenced).
April 19, 2024 - 14:12:20
Django version 5.0.3, using settings 'pwdmngr.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

7. OBSERVATIONS

- **Experimental Setup**

Software Requirements:

- Operating system : Windows XP/7/10.
- Coding Language : Python, Django
- Tool : Visual Studio
- Database : PostgreSQL

Hardware Requirements:

- System : Pentium IV 2.4 GHz.
- Floppy Drive : 1.44 Mb.
- Ram : 1 GB

Parameters with formulas:

1. **Device-enhanced Password Authenticated Key Exchange (DE-PAKE) Model Parameters:**
    - *P*: User's master password.
    - *S*: Stored information.
    - *K*: Secret key derived from the user's master password and the stored information.
    - *f*(): One-way function used in the DE-PAKE protocol.
2. **Oblivious Pseudo-Random Function (OPRF) Scheme Parameters:**
    - *x*: User's master password.
    - *y*: Randomized password generated by the OPRF scheme.
    - *g*(): OPRF function used for password transformation.

3. **AES Encryption Parameters:**
    - $'P'$: Encrypted password.
    - IV: Initialization Vector.
    - $E()$: AES encryption function.
4. **RSA Encryption Parameters:**
    - $PKpub$: Public key for encryption.
    - $PKpriv$: Private key for decryption.
    - C: Cipher text.
    - M: Message to be encrypted.
    - $RSA()$: RSA encryption and decryption function.
5. **Salted One-way Hash Parameters:**
    - $H()$: Hash function.
    - $salt$: Random value added to the password before hashing.
    - $hash$: Resulting hashed password.

6. **Domain-specific Parameters:**
    - D: Website domain.
    - $fD()$: Function incorporating website domain into password computations.

Formulas:

1. DE-PAKE: $K=f(P,S)$
2. OPRF: $y=g(x)$
3. AES Encryption: $'P'=E(P,IV)$
4. RSA Encryption: $C=RSA(PKpub,M)$
5. Salted One-way Hash: $hash=H(P||salt)$
6. Domain-specific Computation: $fD(x)$

8. SPHNIX vs Other Password Managers

	SECURITY								USABILITY				DEPLOY.	
	S1. Unique-Password-Enforcer	S2. Resilient-to-Phishing	S3. ODA-Resistant-Server-Compromise	S4. Storeless	S5. ODA-Resistant-Device-Compromise	S6. Resilient-Upon-Theft	S7. Resistant-to-MITM-D-to-C	S8. Resilient-to-Physical-Observation	U1. Memorywise-Effortless	U2. Password-Update-Not-Necessary	U3. Scalable-for-Users	U4. Physically-Effortless	D1. Client-Compatible	D2. Nothing-to-Carry
PM0. Password-Only	n	n	n	y	n	y	-	n	y	y	n	y	y	y
PM1. SPHINX														
a) Smartphone Explicit Consent	y±	y	y	y	y	y	y	y	y	n	y	n	n	n
b) Smartphone Zero Interaction	y±	y	y	y	y	y	y	n	y	n	y	y	n	n
c) Online	y±	y	y	y	y	y	y	n	y	n	y	y	n	y
PM2. PwdHash	y	n⊙	n⋄	y	n	y	-	n	y	n	y	y	n	y
PM3. Password Multiplier	y	n⊙	n⋄	y	n	y	-	n	y	n	y	y	n	y
PM4. Passpet	y	n⊙	n⋄	y	n	y	-	n	y	n	y	y	n	y
PM5. Firefox	n	n	n∗	n	n	n	-	n	y	y	y	y	y	y
PM6.1. Client-based Managers	n	n	n∗	n	n	n	-	n	y	y	y	n	n	y
PM6.2. Token-based Managers	n	n	n∗	n	n	n	y†	n	y	y	y	n	n	n
PM6.3. Online Managers	n	n	n∗	n	n	n	y‡	n	y	y	y	n	y	y
PM7. Tapas	n	y	n∗	n	n	n	y•	y	y	y	y	y	n	n

TABLE 8.1 : SPHINX vs. Other Password Managers. "ODA-Resistance" denotes resistance to offline dictionary attacks.

Notes: ± Enhanced by the use of two uniqueness parameters, domain name and OPRF key. They provide higher resistance compared to password-only, but still an ODA is possible after the phishing attack. Unless the user chooses a randomized master password. ∗ Unless the user chooses a randomized password for each account. † Establishment of confidential channel necessary. ‡ Establishment of confidential and authenticated channel necessary. • Establishment of confidential and authenticated channel necessary.

9. CONCLUSION:

Passwords are a “necessary evil”. In this paper, we attempted to respond to the growing security and usability problems with passwords by proposing SPHINX, a cryptographic password manager that can address most security and usability problems with passwords from the client/user side alone (i.e., transparent to most existing web authentication services). SPHINX is a password management approach, built atop an existing oblivious PRF (OPRF) scheme, that transforms a human-memorable password into a random password with the aid of a device without the need to store the passwords on the device. SPHINX offers 12 several key security guarantees, namely, resistance to: (1) online guessing attacks, (2) offline dictionary attacks under server compromise, (3) offline dictionary attacks under device compromise, (4) phishing attacks, and (5) eavesdropping and man-in-the-middle attacks on the device-client channel. SPHINX also boasts to provide almost the same level of user experience as that of authentication using an easy to memorize password. Unlike other password managers, SPHINX perfectly hides passwords and the master password from itself, and thus remains secure under the realistic threat of the compromise of password managers. Also, unlike other password managers, SPHINX does not require a confidential device-client channel. At the same time and like many other password managers, SPHINX can resist online guessing, offline dictionary under web service compromise and phishing attacks. We designed and implemented a smartphone-based instantiation of SPHINX. Our performance and analytical evaluation of this instantiation shows that it is efficient, highly secure, likely simple to use, and easy to deploy in practice.

10. REFERENCES / BIBILOGRAPHY

[1] 1Password: Simple, Convenient Security. https://1password. com/.

[2] Anonymous hackers claim to leak 28,000 PayPal passwords on global protest day. http://goo.gl/oPv2h.

[3] Blizzard servers hacked; emails, hashed passwords stolen. http: //goo.gl/OTNWJC.

[4] Current Android Malware . http://goo.gl/0sWbXz.

[5] Dashlane Password Manager. https://www.dashlane.com/.

[6] Fine the source of your leaks. https://www.leakedsource.com/.

[7] Hackers compromised nearly 5M Gmail passwords. http://goo. gl/IRu07u.

[8] LinkedIn Confirms Account Passwords Hacked. http://goo.gl/ AWB5KC.

[9] Nanohttpd java app. https://nanohttpd.com.

[10] One Year Of Android Malware. http://goo.gl/2UkUJS.

[11] Password Manager - Remember, delete, change and import saved passwords in Firefox. https://goo.gl/Qve4l7.

[12] Password Manager, Auto Form Filler, Random Password Generator & Secure Digital Wallet App. https://lastpass.com/.

[13] RoboForm: World's Best Password Manager. https://www. roboform.com/.

[14] RSA breach leaks data for hacking securid tokens. http://goo.gl/ tcEoS.

[15] RSA SecurID software token cloning: a new how-to. http://goo. gl/qkSFY.

[16] Russian Hackers Amass Over a Billion Internet Passwords. http://goo.gl/aXzqj8.

[17] A. Adams and M. A. Sasse. Users are not the enemy. Commun. ACM, 42(12), 1999.

[18] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In Advances in Cryptology – Eurocrypt, 2000.

[19] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. 2013.

[20] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In Security and Privacy (SP), 2012 IEEE Symposium on. IEEE, 2012.

[21] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In Security and Privacy, 2012.

[22] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Crackingresistant password vaults using natural language encoders. In 2015 IEEE Symposium on Security and Privacy, pages 481–498. IEEE, 2015.

[23] S. Chiasson, P. C. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In Usenix Security, 2006.

[24] I. Dacosta, M. Ahamad, and P. Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In European Symposium on Research in Computer Security, 2012.

[25] W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.

[26] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In Theory of Cryptography. 2005.

[27] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In Proceedings of the ACM Conference on Computer and Communications Security, 2012.

[28] M. Gollam, B. Beuscher, and M. Durmuth. On the security of cracking-resistant password vaults. In ACM Conference on Computer and Communications Security, 2016.

[29] J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In Proceedings of the 14th international conference on World Wide Web. ACM, 2005.

[30] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the passwordonly model. In Advances in Cryptology–ASIACRYPT. 2014.

[31] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. DeviceEnhanced Password Protocols with Optimal Online-Offline Protection. http://eprint.iacr.org/2015/1099. ASIACCS 2016.

[32] A. Karole, N. Saxena, and N. Christin. A Comparative Usability Evaluation of Traditional Password Managers. In International Conference on Information Security and Cryptology, 2010.

[33] A. Karole, N. Saxena, and N. Christin. A comparative usability evaluation of traditional password managers. In Information Security and Cryptology-ICISC. 2011.

[34] M. R. Karthiga and M. K. Aravindhan. Enhancing performance of user authentication protocol with resist to password reuse attacks. International Journal Of Computational Engineering Research, 2(8), 2012. [35] Z. Li, W. He, D. Akhawe, and D. Song. The emperor's new password manager: Security analysis of web-based password managers. In USENIX Security Symposium, 2014.

[36] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In Financial Cryptography and Data Security. Springer, 2007.

[37] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot. Tapas: design, implementation, and usability evaluation of a password manager. In Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012.

[38] R. Morris and K. Thompson. Password security: a case history. Commun. ACM, 22(11), 1979.

[39] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In USENIX Security Symposium, 2005.

[40] M. Shirvanian, S. Jarecki, H. Krawczyk, and N. Saxena. Sphinx: A password store that perfectly hides passwords from itself. In IEEE International Conference on Distributed Computing Systems, 2017.

[41] F. Stajano. Pico: No more passwords! In Security Protocols XIX, pages 49–81. Springer, 2011.

[42] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In USENIX Security Symposium, 2009.

[43] L. Whitney. LastPass CEO reveals details on security breach. https://goo.gl/WyNjjb, 2011.

[44] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. IEEE Security and Privacy, 2(5), 2004.

[45] K.-P. Yee and K. Sitaker. Passpet: convenient password management and phishing protection. In Symposium on Usable privacy and security, 2006.

[46][Characterization and evolution. In Security and Privacy (SP), 2012.