Designing a Secure Password Management System: The SaveAPass Architecture

Keirelys Marlén De Jesús Aponte
Master in Computer Engineering
Advisor: Alfredo Cruz, PhD
Polytechnic University of Puerto Rico
Graduate Project EXPO, May 2025

Abstract *— This project introduces a human-centered, web-based password manager that prioritizes both security and usability. The application is tailored for non-technical users who often struggle with managing numerous complex passwords, leading to unsafe practices such as password reuse or unprotected storage. By integrating modern frontend technologies and a secure cloud backend, the system enables users to store, retrieve, and manage their credentials easily. The solution applies strong encryption techniques and secure user authentication, ensuring that password data remains protected even in the event of a backend compromise. Unlike existing password managers that may be overly complex or insufficiently secure, this application emphasizes simplicity without sacrificing protection. Its design encourages better password hygiene, aiming to minimize data breaches stemming from weak password habits. The project aspires to set a new benchmark in password management by delivering a safe, accessible, and user-friendly experience for individuals across varying levels of technical proficiency.*

Keywords *— Cloud-based Security, Encryption, Password Management, User-centered Design.*

INTRODUCTION

As online services proliferate, individuals are increasingly required to manage numerous complex passwords across various platforms. This growing demand has given rise to a significant usability and security issue known as "password fatigue." Users often resort to insecure practices such as reusing passwords or storing them in plain text files, which increases their vulnerability to cyberattacks. Studies, such as Security Analysis on Password Managers Applications by Alrushaid and Algarawi, emphasize that weak password practices remain a major cause of data breaches [1]. While password managers exist to address this issue, many fall short, either due to their complexity, which deters non-technical users, or due to security flaws, as observed in the Stanford University paper Password Managers: Attacks and Defenses, which highlights vulnerabilities in autofill mechanisms and remote attack surfaces [2].

This project, SaveAPass, proposes a human-centric, secure web-based password-manager designed to simplify and safeguard credential management. The application is built using React 18 for its frontend interface and Firebase for backend services, including authentication and data storage. These technologies are selected to ensure ease of use, real-time responsiveness, and robust cloud support. SaveAPass encrypts user passwords locally using AES through the CryptoJS library before any data is sent to the backend, ensuring that even Firebase cannot access plaintext passwords. This approach addresses the core concern of trust by guaranteeing that sensitive information is never stored in a readable form.

The project also explores current research and best practices to inform its design. Usability and accessibility for non-expert users are prioritized to encourage wider adoption. The development is guided by a series of research questions focused on secure design principles, efficient client-side encryption, and optimized use of Firebase services. As Carr and Shahandashti note, effective security tools must balance strength with usability to be truly impactful [3]. SaveAPass aims to meet this standard by delivering an intuitive interface that simplifies the creation, storage, and retrieval of passwords without compromising security.

However, several challenges must be addressed in this endeavor. Frontend-heavy applications are

susceptible to vulnerabilities such as cross-site scripting (XSS) and man-in-the-middle (MITM) attacks, necessitating strict content security policies and the use of HTTPS protocols as recommended by Mozilla Developer Network and Google Developers [4][5]. Another challenge lies in balancing simplicity with the need for advanced features; too many options may overwhelm users, while too few may limit functionality. Additionally, reliance on third-party tools like Firebase and CryptoJS requires continuous vigilance to avoid inherited vulnerabilities. Transparency in data handling and encryption practices is also essential to build user trust.

Ultimately, SaveAPass aims to create a secure, accessible, and reliable password manager that empowers users, especially non-technical ones, to take control of their digital security. By combining strong encryption, secure authentication, and an intuitive design, this project seeks to set a new standard for password management solutions that are both effective and user-friendly.

REVIEW OF LITERATURE

In today's digital landscape, the growing threat of cyberattacks and data breaches has made strong password practices an essential component of personal and organizational cybersecurity. A recurring vulnerability in many breaches is the reuse of passwords across different platforms. Password managers have emerged as an effective solution by enabling users to store, generate, and manage unique passwords without the burden of memorization.

One influential study, *Password Managers: Attacks and Defenses* from Stanford University, evaluates several popular password managers, primarily focusing on weaknesses in autofill mechanisms. While the SaveAPass project does not incorporate autofill functionality, the paper highlights the necessity of secure system design principle that SaveAPass embraces through encryption-based workflows and robust authentication requirements [2].

Another significant contribution comes from A Study for an Ideal Password Management System [6]. The study advocates for password managers to prioritize both usability and security to increase adoption. It emphasizes features like automated password generation, which reduce reliance on user memory and encourage the use of strong, randomized credentials [6].

In the same vein, *Security Analysis on Password Managers Applications* by Alrushaid and Algarawi underscores how password managers can play a pivotal role in combating password reuse. The study supports the inclusion of tools for generating secure, random passwords as a key strategy for improving password hygiene [1].

While SaveAPass incorporates secure random password generation, this feature is not its sole focus. The application allows users to either generate a random password or manually input their own, depending on preference. What distinguishes SaveAPass is the encryption model: passwords, regardless of how they are created, are encrypted using a secret key generated during the sign-up process. This secret key is itself encrypted with the user's master password and securely stored in Firestore, making it inaccessible to anyone but the authenticated user.

Together, these studies validate the importance of secure password management tools and support the core features implemented in SaveAPass. By not only generating strong passwords but also encrypting all stored credentials with user-specific keys, the project addresses both usability and security in a comprehensive manner.

Development Approach

The development of SaveAPass was guided by the goal of delivering a secure, user-friendly, and scalable password management application. To achieve these objectives, a careful evaluation of various technologies was performed, focusing on frontend frameworks, backend services, and encryption methodologies. Each technology was selected to meet the project's core principles of simplicity, security, and accessibility.

Frontend Framework Selection

Several popular JavaScript frameworks were considered to build the frontend of SaveAPass, with the aim of providing dynamic and efficient user experience. The key frameworks evaluated included React, Angular, and Vue.js [7] [8] [9] React stood out due to its component-based architecture, which enhances reusability and maintainability. Its virtual DOM optimizes rendering performance, and the robust ecosystem of libraries and tools available for React made it an ideal choice. Given its widespread adoption and strong community support, React was selected to power the user interface of SaveAPass.

Backend Services Selection

In evaluating backend options for SaveAPass, several cloud-based platforms known for their capabilities in authentication, data storage, and hosting were considered. The goal was to select a solution that offered strong security features, ease of integration with the frontend, and minimal server management overhead. Table 1 shows the different backend services that were considered, Firebase, AWS Amplify and Heroku, each one of them has features that made them candidates for this project.

Table 1
Backend Services Considered for SaveAPass

Services	Description
Firebase	A platform by Google offers a suite of services, including real-time databases, authentication, and hosting. Firebase simplifies backend development with its serverless architecture and seamless integration with frontend applications [10].
AWS Amplify	A set of tools from Amazon Web Services designed to build scalable applications. Amplify provides features like authentication, APIs, and storage, with robust cloud infrastructure support [11].
Heroku	A platform-as-a-service (PaaS) that enables developers to deploy, manage, and scale applications. Heroku supports multiple programming languages and offers a straightforward deployment process [12].

After comparing these platforms, Firebase emerged as the most suitable backend service for SaveAPass. Its real-time database capability enables instant data synchronization across devices, which is essential for responsive user experiences. Firebase's tight integration with React allowed for faster development cycles, while its serverless nature minimized the need for complex backend infrastructure. Furthermore, Firebase’s robust security features, including authentication and role-based access control, aligned well with the project's focus on safeguarding sensitive user data. These advantages made Firebase the optimal choice for powering the backend of SaveAPass.

Encryption Methodology Selection

To secure user passwords and sensitive data, multiple encryption methodologies were considered. AES (Advanced Encryption Standard), bcrypt, and scrypt were evaluated based on their performance and security characteristics. AES is a symmetric encryption algorithm widely adopted for its security and efficiency. It is suitable for encrypting large amounts of data and is recognized as a standard by the U.S. government [13]. Bcrypt, on the other hand, is a password-hashing function designed to be computationally intensive, making it resistant to brute-force attacks. It incorporates salt to defend against rainbow table attacks and is adaptive, allowing the iteration count to be increased over time [14]. Similarly, scrypt is a memory-hard password-based key derivation function designed to make large-scale custom hardware attacks costly, providing strong resistance to hardware-based brute-force efforts [14]. After careful evaluation, AES was chosen for SaveAPass due to its strong security and high performance, making it ideal for real-time encryption tasks. To implement AES encryption securely within the frontend, CryptoJS was selected as the encryption library. This choice enabled passwords to be encrypted on the client side before being stored in Firebase, adding an extra layer of security. The combination of AES encryption and CryptoJS ensured that even if the database was compromised, the encrypted data would remain protected.

Technology Integration

The selected technologies work seamlessly together to provide a secure, responsive, and user-friendly password management solution. React handles the user interface logic, including input forms and dialogs. When a user enters a password to store, React triggers CryptoJS to encrypt the password using AES. The encrypted data is then securely transmitted to Firebase Firestore for storage. Firebase Authentication ensures secure user sessions, while Firestore rules control access to the data [13][15][16].

When a user wants to view or edit a stored password, the encrypted data is retrieved from Firebase, decrypted using CryptoJS with the correct secret key, and displayed temporarily. This end-to-end encryption process ensures that only the authenticated user can access the plaintext password, even if the database is exposed. By integrating React, Firebase, AES, and CryptoJS, SaveAPass provides a secure and user-friendly experience while maintaining the highest standards of data protection.

In conclusion, the thoughtful selection and integration of technologies have resulted in a password management solution that emphasizes security, performance, and usability. React's flexibility, Firebase's real-time database, and AES encryption collectively form the foundation of SaveAPass, ensuring a smooth and secure experience for users of all technical backgrounds.

METHODOLOGY

The development of SaveAPass followed a structured and flexible methodology aimed at delivering a secure, user-friendly, and scalable password management application. SaveAPass was developed using an iterative, feature-focused approach rather than a rigid sprint-based Agile methodology. The team prioritized flexibility, allowing for continuous testing and improvement. Features were developed one at a time, with each being completed and tested before progressing to the next. This progressive strategy enabled rapid decision-making and refinement throughout the project.

Development was driven by three core practices: feature-based progression, immediate testing after each feature's implementation, and ongoing enhancements based on test results and evolving requirements. This method ensured the application matured organically, with a constant focus on usability and security.

System Architecture

SaveAPass employs a frontend-centric architecture, leveraging React for dynamic user interfaces and Material UI for responsive, accessible design components. The backend infrastructure is powered by Firebase, which provides robust services including user authentication, real-time database management through Firestore, and secure hosting. This architecture is designed with a strong emphasis on security, ensuring that all sensitive data, particularly user passwords, is encrypted on the client side before being transmitted or stored.

The frontend handles critical operations such as user interaction, form validation, password generation, and the core logic for managing stored credentials. It also performs client-side encryption to prevent exposure of plaintext data. Firebase Authentication facilitates secure user sign-up, login, session management, and password reset workflows. Firestore, a NoSQL cloud database, is used to store encrypted passwords along with associated metadata, such as service names, timestamps, and user identifiers [16].

To safeguard data access, Firebase Security Rules enforce strict access controls, ensuring that each user can only read or write their own records. As illustrated in Figure 1, client-side encryption is implemented using the AES (Advanced Encryption Standard) algorithm through the CryptoJS library. Each password is encrypted before transmission to Firestore, ensuring that no unencrypted sensitive data ever leaves the user's device. This approach provides end-to-end confidentiality and minimizes the risk of data exposure even in the event of a

server compromise or unauthorized access to the database.

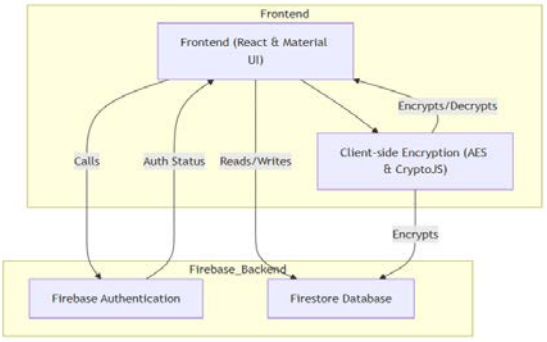

Figure 1
Frontend and Backend Interaction

The encryption flow includes the generation of a unique secret key during user registration, which is encrypted with the user's password and stored securely in Firestore. When users add, view, or edit passwords, the system prompts their password to decrypt the secret key, ensuring that encryption and decryption processes remain private. The key is re-encrypted during password changes, maintaining continuity and security.

Security Implementation

Security is integral to SaveAPass's architecture. Authentication is managed through Firebase, ensuring robust session control and a session timeout mechanism to automatically log out inactive users. Each user has a dedicated, encrypted AES key, stored in Firestore under a secure userSecrets collection.

Data protection is enforced using AES encryption via CryptoJS, applied on the frontend. All data is transmitted over HTTPS, and Firestore security rules are configured to ensure access is restricted to authenticated users only [10]. This comprehensive, layered security model ensures that users' passwords remain confidential and protected from unauthorized access.

User Interface and Experience Design

The user interface of SaveAPass was designed for simplicity and clarity. A minimalist layout ensures ease of use while upholding a professional appearance. Material UI was used to maintain consistency and responsiveness across all screen sizes. Accessibility was a key consideration, ensuring the app could be used by individuals with varying levels of technical experience.

Consistent navigation was implemented across features such as the password vault, adding and managing passwords and settings. The design aimed to reduce cognitive load while still offering all necessary functionality to users in an intuitive manner.

Feature Logic and Code Flow

Each feature in SaveAPass was implemented using modular React components, integrated with Firebase and CryptoJS. During the sign-up process, shown in Figure 2, users fill out a form on the front end. Firebase Authentication securely creates the account, and a unique 256-bit AES secret key is generated using CryptoJS. This key is then encrypted using the user's password and securely stored in Firestore under the userSecrets collection. The entire process is governed by a central function that handles both sign-up and login flows, as outlined in the pseudocode shown in Figure 2.

```
Function generateSecretKey:
    Create 256-bit random key

Function encryptKey(key, password):
    Return key encrypted with password

Function handleFormSubmit:
    Prevent default, show loading
    If signup: create user, encrypt & store key
    If login: authenticate user
    On success: go to dashboard
    On error: show message
```

Figure 2
Sign-Up and Login Process Pseudocode

Figure 3 shows the process of adding a password, the user clicks "Add Password," opening a dialog to input service name, email/username, and password, which the user can create their or generate a password. After submission, the app prompts for the user's account password. This is used to decrypt their AES key. The entered password is then encrypted and stored in Firestore along with metadata. If the decryption step fails, an

error is displayed. This secure process ensures only authorized access to sensitive data.

```
Function generateSecurePassword:
    Create 8-char random password and store it

Function togglePasswordVisibility:
    Switch between hidden and visible password

Function savePassword:
    Confirm save, then decrypt user's key
    If key not found or
        decryption fails: show error, exit
    Encrypt password, save with user data
    Show success, reset form, close dialog

Function cancelSave:
    Go back to 'Password Vault'
```

Figure 3
Add Password Process Pseudocode

When viewing a password, the user clicks the “View” button next to a vault entry. A dialog appears, requesting the user’s account password. Upon entry, the AES key is retrieved and decrypted. If successful, the encrypted password is decrypted and displayed. Passwords are never stored or transmitted in plaintext, and decryption only occurs per view request, ensuring maximum security.

For deleting a password, the user clicks the "Delete" button and is prompted to re-enter their account password. This password is used to reauthenticate with Firebase. If the authentication is successful, the password entry is removed from Firestore and the UI is updated in real time. If not, the deletion is aborted. This ensures that only verified users can delete stored credentials.

Figure 4 illustrates the process of editing a saved password in SaveAPass. When a user clicks the Edit button next to a saved password entry, a dialog appears with the service name and username fields pre-filled, while the password field is intentionally left blank for security purposes. The user can update any of the fields, and upon clicking the Save button, a confirmation dialog prompts them to re-enter their account password. This password is used to decrypt their secret encryption key, ensuring that any updates remain securely protected. The newly entered or modified password is then re-encrypted using this key and saved back to Firestore, replacing the old entry. Once the operation is successful, the application notifies the user and navigates back to the main vault view.

```
On selected password:
    If exists: prefill form, clear password

Function generateSecurePassword:
    Create and store 8-char password

Function handleSavePassword:
    Open confirmation dialog

Function handleDecryptAndSave:
    Get user ID and key, clear errors
    If key missing/invalid: show error, exit
    Encrypt password, save it
    Show success, close dialog
```

Figure 4
Edit Password Process Pseudocode

To change their account password, the user begins by entering and confirming a new password. The application first checks that both fields are filled and that the passwords match. If these validations pass, a dialog appears prompting the user to input their current (old) password. This password is used to retrieve and decrypt the user’s encrypted secret key stored in Firestore. If the decryption is successful, the application proceeds to update the user's Firebase account password using the updatePassword () method. Next, the secret key is re-encrypted using the new password and saved back into the userSecrets collection. This process ensures that previously saved passwords remain accessible, as the underlying secret key used for encryption stays the same, only the password that protects it changes. By re-securing the secret key with the new password, the system maintains both security and continuity for the user.

When the user opts to delete a saved password, they are prompted with a dialog to confirm the action by entering their account password. The application then re-authenticates the user using the provided credentials. If the re-authentication is successful, the app deletes the user account from Firebase using the deleteUser () function. There is no need to decrypt any saved data during this process, as the entire account and all related data, including encrypted passwords and secret keys, are removed from the database. This ensures complete

and secure deletion without leaving any residual information in the system.

RESULTS

The password manager web application achieved its main goal of allowing users to securely manage their login credentials within a clean and intuitive interface. Users begin by creating an account or logging in with an existing one. This process is powered by Firebase Authentication, which ensures that only verified users can access the application. It forms the foundation of security in the system, as each user's data is linked to their authenticated session. Figure 5 illustrates the entry screen where users can sign up for a new account or log in with their credentials.

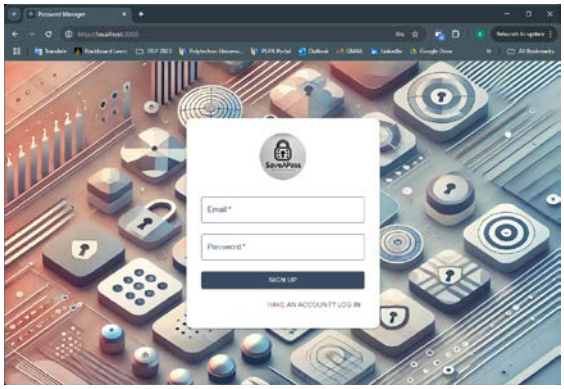

Figure 5
SaveAPass User Sign Up Screen

After authentication, users are taken to their personalized dashboard, which contains the Password Vault. This is the core feature of the application, where users can store credentials for different services such as email, social media, or banking platforms. Users can add new entries, generate strong passwords, and update or delete saved credentials. Importantly, all passwords are encrypted on the client side using AES encryption through CryptoJS, ensuring that no readable data is stored in the database. Each user's encryption key is unique and protected by their account password. Only after re-authenticating with their password can the user decrypt this key to view or update any stored information. Figure 6 illustrates the dashboard where the password vault is displayed, showing the stored entries and available actions.

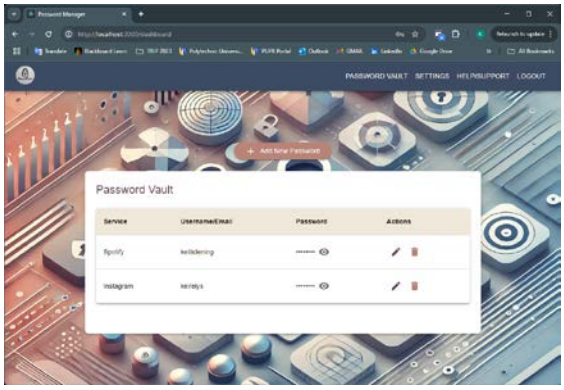

Figure 6
SaveAPass Password Vault Screen

Each time a user performs sensitive actions such as viewing, adding, or editing a password, they are prompted to re-enter their account password. This step requires decrypting the secret encryption key and ensures that the user's identity is verified before any changes are made.

The Settings section allows users to change their account password or delete their account entirely. Both actions involve a confirmation step to verify the user's identity. When changing the password, the encryption key is re-encrypted with the new password to maintain continued access to encrypted data. If a user chooses to delete their account, both their Firebase Authentication profile and Firestore records are securely removed.

The application also includes a Help and Support page to assist users in understanding its features, and a Logout function that safely ends the session by clearing temporary data and redirecting users to the login screen. Altogether, the app emphasizes both usability and security, delivering a reliable platform for password management.

CONCLUSION

SaveAPass offers a secure, intuitive solution for managing passwords in an increasingly digital world. It empowers users to take control of their credentials by combining strong encryption, reliable authentication, and a user-friendly design.

As shown in the interface, the application makes complex security operations accessible through a simple and clean layout.

The decision to encrypt all sensitive data on the client side, using a secret key tied to the user's password, ensures that only the user can access their vault. Even during actions like editing or deleting entries, the app requires password re-authentication to unlock the key, maintaining strict access control throughout.

By replacing insecure browser prompts with custom dialogs, SaveAPass provides a more consistent and secure experience. It shows how strong security doesn't have to come at the expense of usability. Every feature is designed to prioritize user privacy while remaining straightforward to use.

Discussion

The development of the SaveAPass application focused on creating a secure and easy-to-use platform for managing passwords. Its core is a password vault that allows users to store, view, edit, and delete login credentials securely. All data is encrypted on the client side using AES encryption, and a user-specific secret key ensures that only the rightful user can access their information. This secret key is encrypted with the user's password and stored securely in Firestore, never exposed in plain text. When users perform sensitive operations like adding or viewing a password, they are prompted to re-enter their password. This password is used to decrypt the secret key, which is then used to handle the encrypted data.

Changing a user's account password is also handled securely. The user must first authenticate with their current password to decrypt the secret key. Once they enter a new password, the system re-encrypts the secret key with this updated password. This ensures that data remains protected and only accessible to the user, even after password changes.

The interface, built with React and styled using Material UI, focuses on making security processes feel simple and intuitive. Instead of using traditional confirm () or prompt () dialogs, the application uses polished Material UI dialogs for actions like account deletion or password changes. This not only improves user experience but also ensures better security by avoiding browser-dependent prompts.

Behind the scenes, every interaction with the database follows a well-defined flow. On sign-up, a secret key is generated and encrypted with the user's password. When passwords are added or viewed, this key is decrypted using user input and then used to encrypt or decrypt the stored data. The application never stores decrypted passwords or keys, ensuring that even if the database were compromised, the data would remain unreadable without the user's password.

Through this approach, SaveAPass maintains a strong focus on both user experience and data protection, using encryption and thoughtful UX patterns to make security feel effortless.

Overall, SaveAPass is more than just a technical project, it's a real-world tool that reflects best practices in secure software design. It lays a solid foundation for future enhancements like 2FA, logging, backups, and organizational features. With its current functionality and planned improvements, SaveAPass demonstrates how modern applications can strike the right balance between simplicity, usability, and security.

Future Work

While the current implementation of SaveAPass is both functional and secure, there are several areas where future improvements could enhance the application further.

Introducing two-factor authentication (2FA) would provide an additional layer of security. This could involve a one-time code sent to the user's phone or generated by an authenticator app. Even if a password is compromised, 2FA would make unauthorized access significantly more difficult.

The addition of user activity logging would improve transparency and help users monitor access to their accounts. Events such as logins, edits, and deletions could be tracked and reviewed

to detect unusual behavior. Alerts for login attempts from unfamiliar devices or locations would further strengthen account protection.

A data backup and recovery feature would also be valuable. Encrypted backups stored in the cloud could help users recover their vault in case of device loss or accidental deletion, maintaining both accessibility and security.

Improving usability can also enhance security. A password strength checker could guide users to create stronger passwords during the add or edit process. This would help prevent weak credentials from being stored and encourage better password hygiene.

Finally, user-centered additions such as grouping passwords into categories, secure password sharing, or password expiration reminders could increase the practicality of the tool. These features would provide a more organized experience and help users stay proactive with their security practices.

REFERENCES

[1] A. Alrushaid and R. Algarawi, “Security Analysis on Password Managers Applications,” in *Research Publish*, 2020. [Online]. Available: https://www.researchpublish.com/upload/book/paperpdf-1600434399.pdf. [Accessed: Dec. 11, 2024].

[2] D. Silver, S. Jana, E. Chen, C. Jackson, and D. Boneh, “Password Managers: Attacks and Defenses,” in *Stanford University*, 2014. [Online]. Available: https://crypto.stanford.edu/~dabo/pubs/papers/pwdmgrBrowser.pdf. [Accessed: Dec. 20, 2024].

[3] M. Carr and S. F. Shahandashti, “Revisiting Security Vulnerabilities in Commercial Password Managers,” in *arXiv preprint arXiv:2003.01985*, 2020. [Online]. Available: https://arxiv.org/abs/2003.01985. [Accessed: Dec. 20, 2024].

[4] Mozilla Developer Network (MDN). (2025). *HTTPS Overview* [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. [Accessed: Jan. 6, 2025].

[5] Google Developers. (n. d.). *Using HTTPS on Your Website* [Online]. Available: https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https. [Accessed: Jan. 16, 2025].

[6] S. K. Shinde and M. V. Deshpande, “A Study for an Ideal Password Management System,” in *International Journal of Research in Applied Science & Engineering Technology*, vol. 8, no. 4, pp. 123–130, Apr. 2020. DOI: https://doi.org/10.22214/ijraset.2022.39970. [Accessed: Jan. 18, 2025].

[7] Meta Platforms, Inc. (n. d.). *React – A JavaScript library for building user interfaces* [Online]. Available: https://react.dev/. [Accessed: Feb. 14, 2025].

[8] Google LLC. (2025). *Angular – Web Framework* [Online]. Available: https://angular.io/. [Accessed: Feb. 14, 2025].

[9] E. You. (2025). *Vue.js – The Progressive JavaScript Framework* [Online]. Available: https://vuejs.org/. [Accessed: Feb. 14, 2025].

[10] Firebase Documentation. (n. d.). *Security & Rules* [Online]. Available: https://firebase.google.com/docs/rules. [Accessed: Feb. 20, 2025].

[11] Amazon Web Services, Inc. (2025). *AWS Amplify Documentation* [Online]. Available: https://docs.amplify.aws/. [Accessed: Feb. 20, 2025].

[12] Salesforce.com, Inc. (2025). *Heroku: Cloud Application Platform* [Online]. Available: https://www.heroku.com/. [Accessed: Feb. 20, 2025].

[13] CryptoJS. (n. d.). *JavaScript Library of Crypto Standards* [Online]. Available: https://cryptojs.gitbook.io/docs/. [Accessed: Feb. 24, 2025].

[14] OWASP Foundation. (2025). *Password Storage Cheat Sheet* [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. [Accessed: Feb. 24, 2025].

[15] Firebase Authentication. (n. d.). *Firebase Authentication Documentation* [Online]. Available: https://firebase.google.com/docs/auth. [Accessed: Mar. 3, 2025].

[16] Firebase. (n. d.). *Cloud Firestore Documentation* [Online]. Available: https://firebase.google.com/docs/firestore. [Accessed: Mar. 5, 2025].