Sesame: A Secure and Convenient Mobile Solution for Passwords

Mehrdad Aliasgari*, Nick Sabol†, and Ashutosh Sharma†
Department of Computer Engineering and Computer Science,
California State University, Long Beach
Long Beach, CA, USA
mehrdad.aliasgari@csulb.edu*
{nicksabol91,ashutosh.anil.sharma}@gmail.com†

***Abstract*—Passwords are the main and most common method of remote authentication. However, they have their own frustrating challenges. Users tend to forget passwords that are chosen to be hard to guess. Password managers are an approach to keeping our passwords safe. However, they mainly rely on one master password to secure all of our passwords. If this master password is compromised then all other passwords can be recovered. In this work, we introduce Sesame: a secure yet convenient mobile-based, voice-activated password manager. It combines all different methods of user authentication to create a more robust digital vault for personal data. Each password is encrypted with a new fresh key on the user's mobile device for maximum security. The keys are stored in our servers in a protected format. The user has the option of backing up the encrypted passwords in any cloud service. To view a password, the user only needs to utter the name of a web service, and speaker and speech recognition are applied for authentication. Only the key for that service is sent to the mobile application and the password is decrypted and displayed. The biggest advantage of Sesame is that the user need not assume any trust to neither our servers nor any cloud storage. Also, there is no need to enter a master password every time since speaker recognition is used. However, as an alternative to voice, users can view their passwords using a master password in case voice is not available. We provide a brief analysis of the security of our solution that has been implemented on Android platform and freely available on Google Play. Sesame is an ideal and practical solution for mobile password managers.**

***Index Terms*—Mobile Authentication, Password Management, Secure Cloud Storage, Biometrics, Speaker recognition**

I. Introduction

Typically, there are three different methods of user authentication: 1) *"What you know"*such as passwords and PINs, 2) *"What you have"*such as ID cards and token and finally 3) *"Who you are"*such as biometrics including voice, iris and handwriting.

Passwords are the main and most common method of remote authentication. However, they have their own frustrating challenges. In today's connected world, a person is registered in multiple web services, each requiring a specific type of password. Users are asked to avoid common and easy-to-guess passwords [14]. Often the administrators require a combination of letters, special symbols and numbers. This has led users to reuse a few passwords [16]. In fact, the majority of users have three or fewer passwords that they use for all of their online log-ins [15], [7]. This is quite a dangerous practice. If any of those "master" passwords is compromised or shared, then the user has to recover from theft of passwords on multiple services. Unfortunately, those reused passwords have low-entropy and are therefore easy to guess[23]. However, the restrictions on the choice of passwords makes it really hard for an average person to remember all of their passwords.

Password managers appeared to be the solution in this regard. They serve as a digital vault where we can store all of our passwords and never rely on human memory to remember them. This allows us to use a unique password for each web service. Generally, there are three different types of such password managers. The first category are desktop password managers where user passwords are stored locally on the user's computing device. Examples in this category are Mozilla Firefox, Apple MacOS Keychain, RoboForm [21]. These password managers offer no mobility, and in today's world this is a considerable drawback. The second category stores user passwords online on third-party cloud servers (e.g., LastPass [19] and Firefox Sync [18]). They offer mobility, but the user has to trust the third party. Any attack on the cloud servers could be catastrophic to the user. The third category stores passwords on either user phones or dedicated USB devices. KeePass [2] is a well-known and open source example of phone-based password managers. In USB devices, examples include SplashID Key Safe [5] and Victorinox Slim[8]. However, they pose some mobility issues. The USB has to be carried at all times and physically protected. Karole et al.([17]) conducted a study that found users prefer the third category over the online category of password management. They discuss users' inclination towards phone-based password managers.

In all the above categories, the user is asked to set a master password initially. A key is derived using this master password and is used to encrypt all passwords before storage. This, however, poses a serious security threat. If this master password is compromised (e.g., offline dictionary attacks) then all of user's passwords are compromised. Basically, at the bottom line, all user log-ins depend on one single password.This does not provide the security level that we intended to achieve.

In this work, we introduce a novel approach. Our phone-based online password manager combines all three methods of authentication. We call our solution *"Sesame"*in honor of the story of "Ali Baba and the Forty Thieves". Sesame is a phone application that encrypts each user password with a fresh randomly chosen key and stores the encrypted passwords locally and a backup in a cloud of the user's choice. The encryption keys are stored on Sesame servers in an encrypted format. To retrieve the keys, we require user authentication using their voice. Speaker recognition and speech recognition are applied to the user's voice to confirm user authenticity and what they said. Then only the keys related to the service name the user asked for are sent back to the phone. The application displays usernames and passwords only for the requested web service.

Our solution combines the best of all previous approaches. Since each password is encrypted with a new key, users can enjoy the mobility of an online solution with no security compromise. The keys are stored in an encrypted format on Sesame servers and only can be decrypted with a master password. However, the server only releases the keys to Sesame application when the user's biometric (voice) is confirmed. The users enjoy the convenience of speaking to Sesame servers to retrieve a password without typing. We implemented our solution on Android platform and the application is now publicly available at no fee on Google Play [10].

Organization. We organize the rest of the paper as follows: In section II, we discuss our framework and related work. Section III includes details of our proposed solution. In section IV we provide a brief analysis of our solution and we conclude in section V.

II. Preliminaries

Our solution to password managers utilizes personal mobile devices (tablets, phones and smart watches). We note that mobile devices can not only connect us to Internet but also they can capture, process and store personal data. Mobile devices are becoming more powerful and at the same time more ubiquitous. In fact, 1.76 billion people are expected to own and use smartphones by the end of 2014 [6] and nearly half of Americans own smartphones as well as tablets [3]. Despite all the opportunities, on the other hand, mobile devices have their own security issues. More and more sensitive data is being collected and hosted in cloud computing infrastructures. To prevent unauthorized access to such services, users are asked to provide a log-in credential. Almost all such services require a username and password.

Recently, there has been a sharp growth in the use of mobile password managers. As an example, Lastpass and PasswordBox claim to have well over a million user [19], [22]. Users can add/see and delete their passwords on their mobile devices using the password managers applications. However, they all require a master password to derive one single encryption key when it comes to security of their storage. Since all passwords are encrypted with this one key, if the key is compromised (master password is guessed) all of the user's data is compromised. In addition, research has shown that users do not trust cloud-based solutions in general [17].

In our framework, we distribute data storage. User's sensitive data (e.g., passwords) are encrypted with a new key and stored both on the device's storage and backed up in a cloud of user's choice. This information does not reveal anything about sensitive data since no key is reused (providing maximum security). On the other hand, the keys are stored in Sesame servers. The users need not have any trust in Sesame servers since: 1) the keys themselves reveal nothing about user's data and 2) the keys are encrypted with a key that is protected by a key derived from the master password. Therefore, even in the unlikely event that the cloud and Sesame servers collude, no information about user data is revealed without the master password. This distribution of storage provides a very high level of user privacy that will be discussed in Section IV.

In addition, instead of requiring the user to type in their master password every time, we benefit from biometrics. Note that some modalities of biometrics could carry content. For example, voice or handwriting not only could be used as biometrics for user authentication, but they also could be used for user inputs. On one voice sample, we can perform speaker recognition (authentication) as well as speech recognition (user input). Fortunately, all mobile devices have the necessary hardware to capture voice (i.e., a microphone) at no additional cost. It is very convenient for the user to say what they are looking for instead of scrolling through an application. With the growth of smart watches, this advantage is more noticeable to users' experience.

Moreover, in our solution the user is asked to input their master password only once right after the first launch of the application. The master password is taken, and a key is derived and stored locally. From there on, the user speaks their way into Sesame application with no need to provide the master password. Only the device in which the user has input their master password is capable of decrypting the keys sent by Sesame servers provided the user passes an authentication step. This means that our solution combines all three methods of authentication: who the user is (voice), what the user has and what the user knows (mobile device with the correct key derived from master password). The role of a cloud is for backup purposes and enabling the user to run the application on multiple devices.

Furthermore, we have included an alternative method to voice in user authentication. If the user does not remember the name of a service or does not wish to use their voice (e.g. loud background noise) then the user can enter the "peek view" method. This method offers two alternatives: either the user utters a random word prompted on their application (chosen by Sesame servers) to engage in speaker authentication or they can enter their master password. Either way, upon a successful authentication, all the services are displayed to the user and the user selects the one they wish to view. We note that peek view (explained in more details in section III) is intended to be used as an alternative method and voice recognition is the main method of password recovery.

To the best of our knowledge, there is one company that claims to have a voice-activated password manager[20], [1]. However, their product is proprietary, and there is no clear details available. It appears that the user still needs to trust their company in handling their passwords. Contrary, with our solution, the user assumes no trust to any party. Sesame servers only act as key managers to an authenticated user and learn nothing about user's private data.

III. Proposed Solution

As mentioned in the previous section, we use speaker recognition for user authentication. In this regard, the first time that a user installs the Sesame application we ask for a voice sample of 10 seconds for user enrollment. This sample is processed on the server side and a Gaussian Mixture Model (GMM) is derived. A unique user ID (uid) is assigned to the user and sent back to the application. In addition, the application generates a pair of public key, private key (pk, sk) along with a 256bit encryption key (K_e). Then the user is asked to enter a master password (MP). Using a technique similar to PBKDF2 (Password Based Key Derivation Functions version 2) [12] three keys (K_1, K_2, K_3) are extracted. More details on these keys are provided later. The application creates a file ("Keys.txt") and stores $(uid, K_1, \mathsf{Enc}_{K_2}(K_e), \mathsf{Enc}_{K_2}(pk), \mathsf{Enc}_{K_3}(sk))$, where $\mathsf{Enc}_k(x)$ represents symmetric encryption of x under key k. We use AES 256 bits in CBC mode along with a randomly chosen initial vector (iv) for each encryption. This file ("Keys.txt") is stored on the user's cloud. In the application's folder uid, K_1, K_2, K_e, pk and $\mathsf{Enc}_{K_3}(sk)$ are stored. Note that no information about keys is sent to Sesame servers and more importantly K_3 and MP are never stored/transmitted any where.

To compute the key triplet, we use $\mathsf{HMAC-SHA256}$. For the sake of consistency across various architectures, we encode any integer n used in HMAC as 32 bits integers in big-endian representation. Function $\mathsf{INT\_32\_BE}(n)$ achieves this conversion. We set a pseudo random key (PRK) to be $PRK = \mathsf{ComputePRK}(MP, \mathsf{INT\_32\_BE}(uid), c)$, where the description of ComputePRK is provided below. Basically this function is an HMAC based PBKDF2 where MP is used as the key for HMAC and uid is the input message. The output of this function is 256 bits long. This function calculates HMAC of inputs in iterations. The number of iterations used in PBKDF2 is a constant c. We set this constant to be 4096 in our implementation. The bigger value of c, the more time brute force attacks take to recover PRK. However, resources are limited on a mobile device and thus c cannot be too big. Recall that in $\mathsf{HMAC}(k, m)$ the first input is the key to HMAC and the second is the message.

```
$K \leftarrow \mathsf{ComputePRK}(Key, Salt, c)$

1) $K = 0$
2) $Key = \mathsf{SHA256}(Key)$
3) $U_0 = Salt$
4) For $j = 1$ to $c$
5)     $U_j = \mathsf{HMAC}(Key, U_{j-1})$
6)     $K = K \oplus U_j$
7) Return $K$
```

Next, we set the authentication key ($K_1 = \mathsf{HMAC}(PRK, \mathsf{INT\_32\_BE}(uid||1))$), where $||$ denotes concatenation. This key is used to verify whether the input master password matches the original master password. The key encryption key (K_2) is set as $K_2 = \mathsf{HMAC}(PRK, K_1||\mathsf{INT\_32\_BE}(uid||2))$. This key is used to encrypt the key that encrypts all the password keys (before sending to the Sesame servers). Also the user's public key is encrypted with this key and stored in the user's cloud. This key hides the encryption keys (K_e and pk) that are stored in the cloud. Finally, the private key encryption key ($K_3 = \mathsf{HMAC}(PRK, K_2||\mathsf{INT\_32\_BE}(uid||3))$). This is a backup key that is stored nowhere. The user's private key is encrypted with this key at the first sign-up and stored in the application's folder as well as the backup cloud.

For each entry of service name (S), username (U) and password (P), the application stores locally and in the user's cloud $\mathsf{Enc}_{k_i}(S_i||U_i||P_i)$ for a randomly chosen k_i and i represents the index of the entry. Then the application sends to the server S_i and $\mathsf{Enc}_{K_e}(k_i)$ and $\mathsf{Enc}_{pk}(k_i)$. Note that $\mathsf{Enc}_{pk}(k_i)$ is the public key encryption of k_i. In our implementation we used $\mathsf{RSA-OAEP}$ with $\mathsf{SHA256}$ for padding.

To view a password of a particular service name (S), the user clicks on the microphone icon in the center of main interface and utters S. Voice is captured and sent to Sesame servers. Speaker recognition is applied, and if the voice belongs to the voice model of user uid, then it is a successful authentication. Next, speech recognition is applied to recover S from user voice. The server then selects and sends all $(i, \mathsf{Enc}_{K_e}(k_i))$ where $S_i = S$ (in case of multiple usernames for one web service). The application decrypts k_i and displays all $(S_i||U_i||P_i)$. User can edit/delete each $(S_i||U_i||P_i)$. We refer to Appendix A for screenshots of the application in various stages.

In addition, we have included a third type of password view. We call this approach the "peek view". Peek view allows two different cases of password view. The first case is when the user does not remember service names, and they would like to look at all of their service names. Case two is for when voice authentication is not preferred for any reason (e.g., loud background noise). Below we discuss these two cases:

1. Prompted voice recognition:
   In this case, the user is prompted with a random word selected from the server's dictionary. The user has to utter that exact word, and the voice sample is sent to the server. The server performs speaker and speech recognition and if they both match, a list of all of the user's service names are sent to the application. The user selects the service name they wish and a similar set of steps to the above is executed. We choose a random word each time to prevent replay attacks.
2. Master Password (MP) authentication:
   In this case, the user enters their master password MP'. The application then generates $K_1^{'}$=$\mathsf{HMAC}(PRK', \mathsf{INT\_32\_BE}(uid||1))$, where $PRK' = \mathsf{ComputeKey}(MP', \mathsf{INT\_32\_BE}(uid))$. $K_1^{'}$ is then compared to the original K_1 stored in the application folder. If this is a match, then we assume $MP' = MP$ and generate $K_3 = \mathsf{HMAC}(PRK', K_2||\mathsf{INT\_32\_BE}(uid||3))$; otherwise the application prompts the user with a failure message. Then the application requests

all the service names to be sent from the server. The application displays the service names, and the user selects one, S. Then S is sent to the server, and the server sends back all $i, \mathsf{Enc}_{pk}(k_i)$ where $S_i = S$. The application uses K_3 to decrypt the private key sk. Note that $\mathsf{Enc}_{K_3}(sk)$ is stored in the user's cloud as well as on the device. Using sk the application can decrypt all k_i's and follow similar steps to case one to display $(S_i||U_i||P_i)$.

Our speaker recognition is text-independent, and we use the implementation of Alize/LIA_RAL toolkit by the university of Avignon[13]. As for speech recognition, we use the CMU Sphinx toolkit by Carnegie Melon University[9]. Since we use open source voice recognition tools, our accuracy is limited by the building blocks. In speaker recognition, a voice sample is evaluated to the user's model and a Universal Backgroud Model (UBM). The difference of these two scores in the log domain is then compared to a threshold (θ). If the difference is bigger than θ then we consider this sample to be originated from the user otherwise a failure message will be sent to the application. To set the threshold, we collected voice samples of 110 subjects and conducted 23,409 observations (verification tests). We calculated the true positive rate (TPR) and false positive rate(FPR) as functions of the threshold, θ. Figure 1 (in Appendix) illustrates the outcome. Ideally, we should have $TPR \approx 1$ and $FPR \approx 0$. From our experiment we noticed that $\theta = 0.6$ is a suitable choice in our work. However, we are working on techniques to improve speaker recognition accuracy, such as different score normalizations (e.g., [11]). Also, we note that our servers perform no cryptographic operation other than secure communication to the application using OpenSSL([4]). The main computational load on our servers is voice analysis (speaker and speech recognition) per user request. Any improvement in voice computation will result in an overall performance gain in our system.

IV. Analysis of the proposed Work

We begin with an informal security analysis. The exact proofs are not difficult to derive, and due to space limit are left out. To analyze the security of our proposed work, we consider three different types of adversaries:

- Cloud adversary: an adversary who has access to the user's cloud storage.
- Server adversary: an adversary who has compromised Sesame servers.
- Device adversary: an adversary who has privileged access to the user's device.

In the case of a cloud adversary, all information the adversary can see is the content of "Keys.txt" and $\mathsf{Enc}_{k_i}(S_i||U_i||P_i)$. Each k_i is chosen fresh and at random, and the adversary has no information about them. Therefore, the adversary can learn nothing about user's personal data even if they guess the master password, MP.

A server adversary has access to service names and encryptions of the encryption keys (i.e., $\mathsf{Enc}_{K_e}(k_i)$ and $\mathsf{Enc}_{pk}(k_i)$). The encrypted data has no dependency to the user's personal information. Therefore, a server adversary can recover no data about the user's passwords except all the service names. We note that if a service name is sensitive then the user can register the service name with a nickname instead of an actual name. This way, the server adversary does not know what services are being used by the user.

A device adversary has access to all information a cloud adversary has as well as K_2, K_e and pk. This makes a mobile device adversary more powerful than than a cloud adversary. However, similar to the cloud adversary, the adversary has access to no information about k_i's (encrypted or plaintext). As a result, the adversary cannot recover any personal data alone. Note that this case requires device root privileges for the adversary as well as access to the device. Since mobile devices are carried by the user, generally the existence of this adversary is less likely.

A more interesting scenario is when the cloud adversary and the server adversary collude. In this case, if the random key K_e is recovered so will all k_i's, which would lead to recovery of all passwords. The cloud has access to $\mathsf{Enc}_{K_2}(K_e)$. In other words, the recovery of K_2 leads to the recovery of K_e. However, K_2 is constructed from PRK, K_1 and uid. To recover K_2 the adversary needs to perform a brute force attack. The adversary would need to exhaust all possibility for PRK to construct K_2 which is exponentially expensive and not practical. An offline dictionary attack on MP is not feasible either since uid is used as a salt in the computation of PRK (refer to the iterative function of $\mathsf{ComputePRK}$).The adversary would need to compute PRK and K_1 using uid for each guessed value of MP.

Finally, if a device adversary colludes with a server adversary then the adversary can use K_e stored in the device to recover all k_i's. Each k_i will decrypt a tuple of (S_i, U_i, P_i), and all of the user's personal data will be recovered by the adversary. Only in this case can the system be compromised. However, this type of attack is extremely unlikely since the adversary needs to breach Sesame servers and have root access to the user's mobile device. Even if Sesame servers are breached, this attack will not occur so long as the user keeps their mobile device protected. If the device adversary does not have privileged access to the mobile device then the adversary has to either impersonate the user's voice or guess their master password to recover one password at a time. The main security advantage of our system is that the user need not trust other parties (their cloud services and Sesame servers). Yet they can enjoy having secure and convenient access to their personal data on any mobile device. This is critical as nowadays users are trusting third-party service providers less than before. The users can change their master password at anytime without changing any information about their encrypted private data. We recall that in our implementation we protect usernames and passwords of a user. Nevertheless, this technique can be applied to cloud storage of any type of data in general.

As for system performance and user experience, we do not have a comprehensive set of data yet. This is due to the very recent launch of the application. However, we intend to include an extensive performance and user experience analysis in the journal version of this work. We will extend this work to more platforms with the same user interface in future.

V. Conclusion and Future Work

In this work, we treated the problem of secure password managers. Password managers are more common than before and enable us to choose strong passwords for each of our web services. We develop and implement a convenient yet very secure digital vault that can protect users' private data. Improvements in user experience and performance are future work. In addition, integrating another biometric modality (handwriting) is left for future work. The idea presented here can be generalized for secure cloud storage of any type of data. This generalization is left as future work.

References

[1] Cookeyah. available online at. http://virtualwallet.cookeyah.com/.

[2] Keepass. available online at. http://keepass.info/download.html, 2014.

[3] Mobile technology fact sheet. available online at. http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/, 2014.

[4] Openssl project. available online at. https://www.openssl.org/.

[5] Splashid key safe. available online at. http://www.splashdata.com/splashid/keysafe/, 2014.

[6] Worldwide smartphone usage to grow 25% in 2014. available online at. http://www.emarketer.com/Article/Worldwide-Smartphone-Usage-Grow-25-2014/1010920, 2014.

[7] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[8] Victorinox Swiss Army. Victorinox slim. available online at. http://www.swissknifeshop.com/victorinox-slim-flash, 2014.

[9] Speech at Carnegie Mellon University. Open source toolkit for speech recognition. available online at. http://cmusphinx.sourceforge.net/wiki/.

[10] Security Lab at CSULB. Sesame. available online at. https://play.google.com/store/apps/details?id=net.sesamepass&hl=en.

[11] Roland Auckenthaler, Michael Carey, and Harvey Lloyd-Thomas. Score normalization for text-independent speaker verification systems. *Digital Signal Processing*, 10(1):42–54, 2000.

[12] RSA Laboratories B. Kaliski. Pkcs #5: Password-based cryptography specification version 2.0. available online at. https://www.ietf.org/rfc/rfc2898.txt, 2000.

[13] Jean-François Bonastre, Frédéric Wils, and Sylvain Meignier. Alize, a free toolkit for speaker recognition. In *ICASSP (1)*, pages 737–740, 2005.

[14] Eran Gabber, Phillip B Gibbons, Yossi Matias, and Alain Mayer. How to make personalized web browsing simple, secure, and anonymous. In *Financial Cryptography*, pages 17–31. Springer, 1997.

[15] Shirley Gaw and Edward W Felten. Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security*, pages 44–55. ACM, 2006.

[16] J Alex Halderman, Brent Waters, and Edward W Felten. A convenient method for securely managing passwords. In *Proceedings of the 14th international conference on World Wide Web*, pages 471–479. ACM, 2005.

[17] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A comparative usability evaluation of traditional password managers. In *Information Security and Cryptology-ICISC 2010*, pages 233–251. Springer, 2011.

[18] Mozilla Labs. Firefox sync. available online at. https://www.mozilla.org/en-US/firefox/sync/, 2014.

[19] LastPass. Lastpass password manage. available online at. https://lastpass.com, 2014.

[20] Lavish. Cookeyah: A unique voice biometric free password manager for windows. available online at. http://www.thewindowsclub.com/cookeyah-voice-password-manager.

[21] Siber Systems. Roboform password manage. available online at. http://www.roboform.com, 2014.

[22] R. Woodbridge. How passwordbox passed gmail as the #1 productivity app on their way to over 1m downloads. available online at. http://untether.tv/2013/episode-467, 2013.

[23] Jeff Jianxin Yan, Alan F Blackwell, Ross J Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security & privacy*, 2(5):25–31, 2004.

Appendix

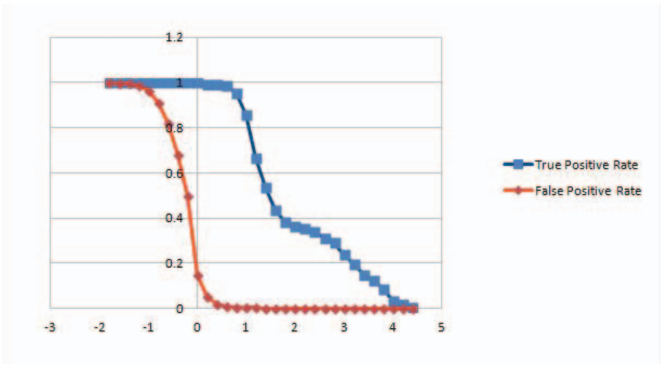
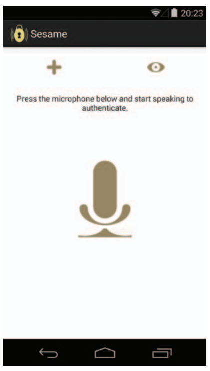
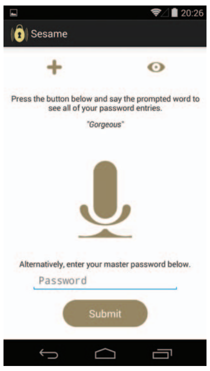


Fig. 1. TPR and FPR as functions of θ (x-axis). $\theta = 0.6$ appears to be a reasonable choice.



Press the microphone below and start speaking to authenticate.

Fig. 2. The main interface of the application. The plus button is for adding a new password entry. The eye button is for "Peek View," and the microphone in the center is to speak to the application.



Press the button below and say the prompted word to see all of your password entries.

"Gorgeous"

Alternatively, enter your master password below.

Password

Submit

Fig. 3. Interface for "Peek View". The user is prompted with a random word, gorgeous in this case. In the alternative, the user can enter their master password to see all of their service names.