An Overview and Analysis of Voice Authentication Methods

Annie Shoup, Tanya Talkar, Jodie Chen, Anubhav Jain
ashoup, tjtalkar, jodiec, ajain94

***Abstract*—Current solutions for passwords and authentication are insecure and have been hacked very regularly. As more and more information is on the Internet, the need for secure authentication has become ever more necessary. Biometric based authentication provides a promising solution because voice biometrics, like fingerprints, create unique identifiers for individuals. We present an analysis of current voice biometric software, propose a security policy for a voice authentication system, and provide two different implementations of potential voice authentication systems that aim to fix some of the deficits of the available open source solutions. In addition, we provide an evaluation of the advantages and disadvantages of our own implementations. Our code is available in a public repository at https://github.mit.edu/jodiec/6857 and https://github.mit.edu/ashoup/dejavuDemo.**

I. Introduction

Over the past few years, it has become increasingly more popular to use voice recognition for applications. Applications such as Google Now and Siri are able to transcribe audio and understand what a user is saying. Some of these applications are also able to uniquely identify the individuals. At the same time, applications such as Shazam and projects like MusicBrainz have been able to identify music using only 2-3 seconds of recorded music. Audio processing has traditionally been too computationally expensive to be able to be done locally–however, that is changing as new processing algorithms are being created. New initiatives are also providing clever ways of using audio for two factor authentication. SlickLogin (acquired by Google) enabled two-factor authentication using audio by playing almost inaudible sounds through a computer's speakers and using a smartphone app to listen to the sound, analyze it, and verify it to the server. The sounds generated were unique to each user and secure so that they could not be played back again later.

As voice-based authentication has evolved, two different approaches to authentication have emerged. The first approach is to have an individual repeat the same sentence multiple times and create a very general template made up of the range of voice prints. When the user speaks in the future, the new voice print generated can be matched to their old voice prints–authenticating the user. However, the downside to this approach is that the voice print is more generic and thus can't be authenticated at the same level of accuracy that voice prints generated with a single passphrase are. This is the second approach–creating a voice print using a single phrase or word and only storing that individual voice print. However, when this approach is used, a third party can record the authentication attempt and replay it to gain access. We explored both of these methods as a part of our survey of available voice authentication techniques. We describe in the next few sections our proposed approach that we hope will accurately identify individuals through unique voice prints. We also describe a prototype that we have built in section VI.

II. Motivation

Bonneau et al. [2] discuss the need to find a replacement for passwords. They provide significant depth into the analysis of biometrics as an alternative for passwords. As more and more information and processing is moved to the cloud, there is an urgent need for even more secure methods to authenticate users. Apple introduced and popularized fingerprint scanning as an alternate method of being able to unlock iPhones, severely hindering the ability to be able to commit any harm with stolen cell phones. The major benefits of biometric data are that it is hard to replicate and that it is unique. However, while voice biometrics are unique, there can be greater risk associated with using them in lieu of passwords. Mainly, voice biometrics can't be changed like a password can, so if they're leaked, there are many more serious consequences [19][20]. Thus, we believe that voice-authentication may not be good as a standalone method for authenticating users, but will be a good option as part of a two-factor authentication protocol. This way, even if passwords

are obtained, the attacker will not be able to bypass the biometric scan and access the user's data.

Along with wanting to come up with a method for making more secure user login, we explored what the current landscape looks like for voice authentication. There are a few different services provided by companies like Nuance, ArmorVox, Authentify, and SpeechPro, however, these were all commercial applications and thus did not provide access to the internal models used to store voice prints. There were few open source options that allowed developers to drop in voice authentication into their application and so we sought to create a solution that provided a method of providing two-factor authentication using voice biometrics as the second form of authentication. Since we found deficiencies with several of the open source implementations, we also tried to implement our own voice authentication system (section VI) as a proof of concept that solves some of the security issues in the open source versions.

III. Security Policy and Threat Model

Before the analysis of already existing open source solutions and the implementation of our voice authentication system, we defined a security policy and threat model which outlined the scope of our project and guided our decisions.

A. Voice Authentication System

The authentication system consists of a client side application and a server. The client side application will reside on a party's device, whether that be a mobile device, personal computer, or tablet. The system will need access to the recording mechanism on the device on the client side and be connected to the Internet. The client side application will make API calls to the server to authenticate users, sending over the recorded audio files over the Internet. The server will analyze the audio, fingerprinting it and determining whether or not the user is the one that he or she is trying to authenticate as. Due to imperfections in voice fingerprinting, the system is intended to be used as a second factor in two-factor authentication. The server stores the voice features associated with a user's account through an enrollment phase where multiple audio samples from one user are utilized to extract features and generate a general voice fingerprint for a user.

B. Person of Interest

The person of interest is a user who has or will create an account in the voice authentication system.

If the person of interest has an account in the voice authentication system, the person of interest may log into their account using voice authentication. In the solutions we have analyzed and in our implementation, we assume that the person of interest has successfully provided the first factor of two-factor authentication. The person of interest is the only person who will be able to log into their account. They do not need to own the device with which they are using to log in. This person will state a phrase given to them by the voice authentication system, and will be told whether or not their voice belongs to the account they are trying to log into.

The person of interest may also enroll in the voice authentication system if they do not have an account in the system. the person of interest should be the only person that will be able to enroll their voice features in the voice authentication system. A person of interest will enroll in the voice authentication system by saying phrases, which will then be recorded and the features of the audio are then analyzed in the system. By enrolling in the system, the system also creates an account for the person of interest associated with those features. In the case of two-factor authentication, these features are stored in the previously made account associated with the user. In order to log into the system, the person will attempt to authenticate themselves by speaking a new phrase provided to them. A person successfully authenticates themselves when the features of the voice file match the features of the voice that is recorded in the system from enrollment.

C. Attacker

An attacker will attempt to gain access to the person of interest's account. We define that, using our system, the attacker will not be able to use their own voice to gain access to the account, as the voice has to match what has been recorded in the system from enrollment. The attacker should also not be able to replay a recorded version of the voice of person of interest to gain access to the system without having that person present.

Since our system design is not intended for building a secure server, we assume that the attacker does not have the ability to bypass the voice authentication system and gain access to the model used to authenticate. Thus, we assume that the attacker will not be able to construct a set of features which will pass the authentication system by targeting the model used.

IV. Analysis of Current Available Options

We explored three different speaker recognition and audio fingerprinting options: Microsoft's Speaker Recognition API [10], Bob [1], and Dejavu [4]. We analyze Microsoft's API and Bob in this section and explore the security of Dejavu in section V.

A. Microsoft Speaker Recognition API

One of the libraries we explored was the Microsoft Speaker Recognition API. This is one of the publicly available APIs that advertises verification and identification. The API requires the creator of the web application to sign up for an account, after which they receive a private key that they include in their request header. To enroll a user into the system, there needs to be 60 seconds of speech, excluding any silences. To authenticate a user, the system takes in any audio of a user speaking.

This API is extremely accurate and is able to strip out most background noise so the user can be in a noisy environment. The application is also computationally fast, so the user does not have to wait long before they receive a response about their authentication.

The API does have some fall backs, however. First, the enrollment phrases are all standard, and are provided on the website, so an attacker can know which enrollment phrases might be used by a user and can pre-record these phrases. The phrases can be played back if an attacker is attempting to hack into an account, and the service will authenticate the attacker as there is no specific audio passphrase needed for authentication. Since the enrollment phrases are set, an attacker can also record the enrollment phrase and create another account with the same voice fingerprint as the user, leading to potential difficulties in authentication.

Secondly, the API request can be intercepted, and the contents can be replaced, since there is no hashing done to sign the request and the private key is provided in the header as plaintext and could be extracted. The attacker can then put in their own recordings for enrollment and authentication. The API also doesn't necessarily need real time recording of a passphrase, and can also take in a pre-recorded sound file. This means that an attacker does not need to replay the audio to authenticate or enroll, but can also just submit the recorded audio. Though developers can get rid of this feature when implementing the API, it still causes a security risk if the attacker gets access to that recording or uses a recording to attempt to log in.

The service is also pay-to-use depending on how many requests are received, which may not be sustainable for large scale systems with many users. It is also not open-source, and therefore if any parameters need to be tweaked by the developer, then they won't be able to make those changes as needed.

B. Bob

We also attempted to use Bob, a signal-processing and machine learning toolbox developed by the Biometrics group at the Idiap Research Institute. This library has been used in other speaker recognition applications, and therefore we had believed it might help in the creation of our application. However, we found that there were many incompatible dependencies, and the dependencies that could be installed took up a lot of memory on our devices. Looking at the voice corpus that was provided, we would have had to process any user's recording with many filters before getting it into the desired format. Consequently, running any experiments for speaker recognition would take computationally longer than we wanted. Due to these constraints, we decided against using Bob for our speaker recognition application.

V. Usage of Audio Fingerprinting

After looking at a variety of potential voice authentication libraries, the python audio fingerprinting library, *Dejavu* [4], seemed to be the best existing library to further explore due to its lightweight nature, easy usability, and accuracy. Dejavu uses the process of audio fingerprinting to match audio samples. Audio fingerprinting involves taking fast Fourier transforms in overlapping windows over the length of the audio sample to create a 2D array spectrogram, in which amplitude is a function of time and frequency. From this spectrogram, corresponding frequencies and times of amplitude peaks are extracted. Finally, the peak frequencies together with the time differences between the peaks are hashed. This hash is used to represent a unique audio fingerprint for a small window of the sample. Thus each sample is represented by a number of audio fingerprints. These fingerprints for an audio sample are then compared to existing fingerprints in a database to identify the most likely existing sample that the test sample is from. The technique of audio fingerprinting is commonly used in music recognition apps such as Shazam and SoundHound.

With this in mind, there are a few obvious advantages to using Dejavu for voice recognition. Dejavu is extremely accurate in matching audio samples even

with varying levels of background noise. Additionally, Dejavu is able to correctly match relatively short audio samples. Compared to other libraries that require 30 seconds of voice recording, Dejavu is able to achieve perfect accuracy with only five seconds of voice recording. All of these factors contribute to a fast, easy, and accurate voice recognition tool.

However, with these advantages also come disadvantages. Specifically, Dejavu's accuracy comes from matching audio samples, not voice samples. Consequently, the enrollment and authentication phrases for a user must be the same. This makes our system vulnerable to replay attacks in which an adversary could record the user during the enrollment or authentication phase and then replay this recording to successfully log into the system. Additionally, because a user's voice fingerprint is very specific, any decrement or change, such as a cold, to a user's voice might leave the user unable to successfully log into the system.

While Dejavu presented audio fingerprinting instead of voice fingerprinting, we wanted to initially pursue that route as it provided us with the ability to create an initial demo of our system. We built a simple Web API wrapper around Dejavu that provided the ability to register new users and then provide authentication for users. We envisioned our implementation to serve as the second authentication method in a two-factor authentication system. Thus, we created a simple web application which required both a password and a voice sample to login, using the Dejavu API that we built.

Our web application is accessible at 128.30.31.185:8080, screenshots from the web application can be seen in Figure 1. The code for the Dejavu wrapper is available in the server.py file at https://github.mit.edu/jodiec/6857 and the frontend application code is available at https://github.mit.edu/ashoup/dejavuDemo. The Web API provides two different methods, `/register` and `/authenticate`. The register endpoint supports POST requests with four parameters, username and three files. These files are used to train Dejavu on the user's voice and we ask for three samples of the same passphrase. The `/authenticate` endpoint supports POST requests with two parameters, username and a single voice file with the same passphrase. Our API expects the end user application to store which words were used to log the user in so that they will repeat the same passphrase again in the audio sample. This endpoint returns the confidence level and match if the voice sample is matched with a user's previous voice sample, or otherwise returns a Failure which can prompt the end user application to try and let the user re-run the voice authentication.

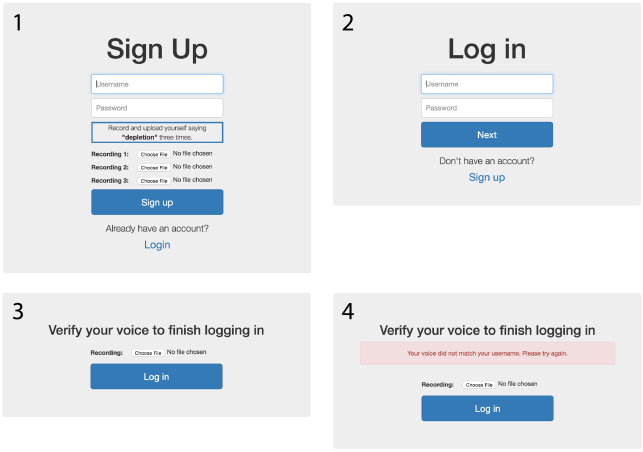

Fig. 1. (1) The screen for signing up. A user enrolls by entering in a username and uploading audio files. (2) The log in screen, a user enters in a username and password. (3) If the password is correct, the user then uploads a recording of them saying their passphrase. (4) If the voice fingerprint does not match, then the user is unable to log into their account.

While this worked, it presented a very brittle solution to our problem as the voice authentication was more audio verification. It also was vulnerable to replay attacks as mentioned earlier. We wanted to explore alternative solutions that provided more robust voice fingerprinting.

VI. GMM Implementation

We decided to implement our own voice authentication system as a proof of concept, utilizing a Gaussian Mixture Model (GMM) to cluster based off of features extracted from audio spectrums.

In our implementation, the process for creating an account is (Figure 2):

1) Enroll user through randomly generated words
2) Speech recognition to make sure user's words match the generated words
3) Attempt voice activity detection
4) Extract and normalize MFCC features
5) Train Gaussian Mixture Model

The process for authenticating an account is similar as we perform similar signal processing (Figure 3):

1) Ask client for the username of the account they want to log into
2) Ask client to speak randomly generated words
3) Speech recognition to make sure user's words match the generated words
4) Attempt voice activity detection
5) Extract and normalize MFCC features

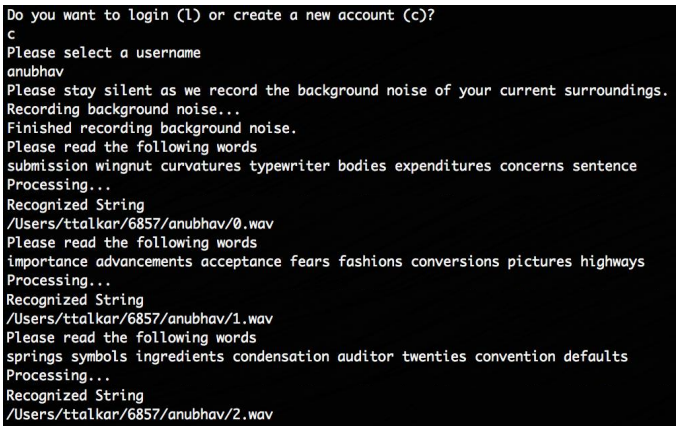

Fig. 2. A screenshot taken of our enrollment process. A user says three different phrases of 8-10 random words.

6) Find best log-likelihood out of all the Gaussian Mixture Models
7) Log the user in if the Gaussian Mixture Model with the best log-likelihood belongs to the account that the client is attempting to log into

Our own implementation of a voice authentication system attempts to address the deficiencies in several of the voice authentication methods that we have analyzed in section IV and addresses our threat model. We decided to do this by (1) enrolling users through randomly generated words and (2) authenticating users through randomly generated words. By doing this, we mitigate the effects of an attacker having access to some number of voice clips of a potential user. We do not depend on a single spoken passphrase and instead rely on a user's unique voice features just as Dejavu [4] does. By requiring users to authenticate by speaking randomly generated words, that prevents users from being logged in just by their voice alone like the Microsoft Speech Recognition API [10]. Similarly, by enrolling through randomly generated words, we address our concerns about Dejavu and Microsoft Speech Recognition API's vulnerability to an attacker having access to a user's voice recordings for enrollment phrases.

When classifying a user attempting to log in, the features for the newly recorded audio are compared to the user's cluster to ensure that they are similar. The resulting application is a command line tool that allows users to either log into an account or to create an account. Our tool successfully authenticates users based on their unique voice features. The code for our command line program is available in the public repository at https://github.mit.edu/jodiec/6857. Our hope is that the tool we built can be successfully built upon by others as the second part of a two-factor authentication scheme.

A. Rationale Behind GMM

We used a GMM instead of other machine learning algorithms because, as indicated in the work done by Reynolds et. al. [15], it is especially useful in speaker recognition. The model combines multiple Gaussian distributions, assuming that feature vectors are independent. A model is computed for each speaker by extracting features out of initial phrases given to the speaker. When the speaker attempts to log in, features are once again extracted from a phrase that is provided to them. Using an equation for log likelihood, we calculate the likelihood that the features extracted belong in the model of the user attempting to log in, or whether there are other models that better fit these features. If the features match the model of the speaker attempting to log in, we can conclude that the speaker is correctly logging into their own account.

B. passphrase

Rather than a traditional passphrase that is unique to every user, we randomly generate passphrases each time a user attempts to log in. Using the python library *RandomWords* [14], we generate between 8-10 random words every time a user has to say a passphrase. Then, using the python library *SpeechRecognition*, we extract the words that the user has said using the Google API integrated into *SpeechRecognition* [18]. This is used to make sure that the speaker is actually repeating the words that were provided to them. Using the Python library *FuzzyWuzzy* [6], we fuzzy match the words generated by the Google API to the words we expect. If the fuzzy matching score is greater than 85, computed by the Levenshtein distance, a way to measure the distance between two strings [7], then we accept the passphrase. If not, then we generate a new passphrase for the speaker to recite, and will keep generating until they are able to say the correct words. In addition, if the speaker does not start speaking within one second of when the words appear on the screen, then the recorder times out, and we generate a new set of random words.

C. Enrollment

When a user decides to enroll into the system, they must first provide a unique username. Then, background noise of their current location is recorded for filtering purposes (subsection VI-D). The user is then

```
Do you want to login (l) or create a new account (c)?
l
username: anubhav
Please stay silent as we record the background noise of your current surroundings.
Recording background noise...
Finished recording background noise.
Please say the following words:
basins minds vicinities licenses clothes barrels halyard bunks minuses
Processing...
basins minds vicinities licenses clothes barrels halyard bunks minuses beeson's mind facilities licenses clothes barrels howard dunks
 minuses
86
max_ltsd = 11.4102047738
lambda0 = 12.5512252512
lambda1 = 25.1024505024
[(7168, 180224)] 185344
tanya -20955.3717869
jodie -22485.9479778
anubhav -14385.8987371
annie -20895.0283333
-14385.8987371
anubhav
Yay! You are now logged in
```

Fig. 3. A screenshot taken from our login process. After creating an account, Anubhav is attempting to log in with his voice. Here we have printed out the speech recognition results, the fuzzy matching ratio, the existing accounts, and the log-likelihood associated with each.

The string was ”basins minds vicinities licenses clothes barrels halyard bunks minuses” and python Speech Recognition interpreted it as ”beeson’s mind facilities licenses clothes barrels howard dunks minuses.” While not exactly the same as the passphrase, the fuzzy ratio of 86 means that the phrase Anubhav said was similar enough to the passphrase that it is likely that he read it. Also, as we can see, even if Anubhav had given a different username, the log-likelihood associated with his account is higher than all the other log-likelihoods, so the command line tool would not have let him access the other accounts.

asked to recite three randomly generated passphrases of 8-10 words,as described above. Once the three passphrases have been recorded, they are filtered and models of their features are created.

D. VAD and LTSD

We want to ensure that we remove as much background noise from the recordings we take as possible. To achieve this voice activity detection (VAD), we take a five second recording of the background noise of the speaker’s location. This becomes our baseline for any noise in the passphrase recording.

From there, we use the long-term spectral divergence (LTSD) algorithm [13] to detect envelopes of areas where the speaker is saying the passphrase words, while leaving out all areas of noise. We accomplish this by using the LTSD algorithm from X. Zhou et. al. [21]. By finding appropriate parameters based off of the background noise, we are able to create envelopes that detect where the speaker is speaking and where there are silences. Figure 4 below is an example of the envelope generated from one of the passphrase recordings. This LTSD generated envelope will be used to extract the signal that is relevant and then passed through the MFCC algorithm to extract features.

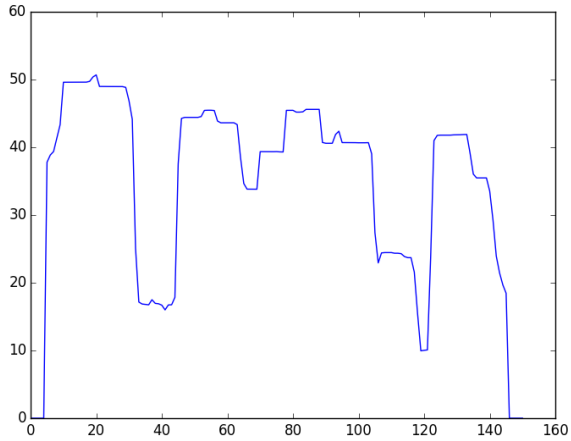

Fig. 4. This is an example of the envelope generated from a passphrase recording. The spikes in the graph correspond to peaks where the user has spoken a word

E. MFCC

The Mel-Frequency Cepstral Coefficient (MFCCs) represents the short-term power spectrum of a sound. It is used in Automatic Speech Recognition as a technique to extract features from audio. MFCCs are the most used features in speech recognition features and are also present in voice fingerprinting implementations [16][21].

First, the signal is split into short time frames. For each of these windows, we take the Discrete Fourier Transform. The powers of this spectrum are mapped onto the Mel scale, a logarithmic curve that models pitches that are typically heard as equal in distance from each other. We take the log of the powers at each of the mel frequencies, and perform a discrete cosine transform. The features that we extract, or MFCCs, are the coefficients of the spectrum that we get from the cosine transform [9].

We extract these features using *python_speech_features* [12]. These are the features we use in our Gaussian Mixture Model after normalization using the mean and inverse standard deviations.

F. Gaussian Mixture Model

To authenticate, we create a Gaussian Mixture Model (GMM) for each user. A GMM is able to combine multiple Gaussians together to form a distribution which will reflect the likelihood of a feature vector falling in a specific area, where the probability will never be zero. The probability that a feature vector x belongs to a model with K Gaussians is:

$$p(x|w_i, \mu_i, \Sigma_i) = \sum_{i=1}^{K} w_i \mathcal{N}(x, \mu_i, \Sigma_i) \qquad (1)$$

where $\mathcal{N}(x, \mu_i, \Sigma_i)$ is a normal distribution with mean μ_i and variance Σ_i [15][21]. The w_is affect the weight that each Gaussian has on the model, and the sum of the w_is must equal one.

To create a model for each user, we take the features from the three recordings and change our parameters–w, μ, Σ–to maximize the log of the likelihood, or probability, that the feature vectors of the recordings belong to the model with those parameters. This is done by calling `fit` in Sci-kit learn's GMM implementation [17].

Once we've created the model, a user can attempt to log in. After extracting the features, the log-likelihood is computed for the features of the recording by using `score_samples` in Sci-kit learn's GMM implementation. `score_samples` returns the per-sample likelihood of the data under the model [17]. We take the model which yielded the highest log-likelihood. If this model matches with the account that the user was attempting to log into, then we return that the user has successfully logged in and can access their account. Otherwise, we return that the user is trying to log into an account that does not belong to them. We were unable to implement a threshold for the log-likelihood to detect when an attacker is attempting to log into an account and does not own one. This is because noise in the authentication recording effects the values of the log-likelihood, so we were unable to find a threshold that worked in all cases. Possible improvements we can make to this are addressed in section VII.

G. Signing and Authentication

One of the potential attacks against the system is a man-in-the-middle attack. An attacker may listen on an unencrypted network and possibly relay information to either the client or the server. In our use cases, the man-in-the-middle attack we are most concerned about is the possibility of the attacker, Eve, sending the server her own audio files in place of a user's. If Eve knows the random words generated by the server, intercepts the user's audio files, and then sends her own audio files to the server, then the user will not be able to access his or her account. Eve can prevent a user from properly creating an account if this is done during the enrollment process as the GMM will be trained on Eve's audio files. We protect against this attack by sending over an audio file to the server as well as a SHA256 hash of the audio file signed with the client's RSA private key. The server can verify if the SHA256 hash of the audio file received is the same as the one included in the signature. While accounting for certificates was not implemented, we can imagine that the public key of a user is verified by a trusted certificate authority. Thus, Eve cannot alter the audio file. If Eve alters the audio file, then the signature will not match. If Eve alters both the audio file and the signature, then the public key will not match the user's certificate.

We are not as concerned with Eve listening in on the network for a user's audio files as the string of random words and their unique ordering results in a huge computation space. Even if Eve records users saying all potential words, it will require a lot of computational power to find the right audio files to string together. For most use cases, this is not an issue. However, our implementation prevents against this attack in our prototype by using RSA encryption and symmetric key encryption [5].

When implementing, we found that we cannot just use RSA to encrypt the audio file bytes as the length of the audio file data is too long for the RSA key size. Knowing this, we decided to use a combination of both RSA encryption and symmetric key encryption. When a client desires to log in, the server knows of the client's public key. The server then generates a

new symmetric key and sends the client the generated symmetric key encrypted with the client's public key. Clients, when attempting to log in, send over a version of the audio file's bytes encrypted with the symmetric key (which they obtained by decrypting with their RSA private key). The server then decrypts the audio file with the symmetric key. Thus, the audio file passed along with the signature, mentioned in the man-in-the-middle attack, is more accurately an encrypted audio file.

VII. Future Work

While we have created a working voice authentication system, there are many improvements that can be made to this system and to our voice authentication security policy such that this method can reliably be used in two-factor authentication.

A. GMM Improvements

When a user attempts to log in, features extracted from the voice recording are used to compute the log likelihood of obtaining those features for every model that is currently stored on the system. For a system which contains a large number of users, the computation time for this could reach a point where the latency of logging in becomes too high for the user. It could also lead to potential attacks if an attacker creates a bunch of dummy accounts just to introduce a larger computation time, thereby affecting anyone who tries to log into the system. As a potential solution, we would like to look at modifying our GMM model to be able to accommodate for clusters, combining with the concept of K-means. Once we know which models would most likely have a high log likelihood for the features we are looking at, based off clustering of the GMMs, we could then only analyze a subset of the GMMs, thereby reducing computation time and preventing any latency attacks. This will make our system more scalable as well.

B. Replay Attacks

Even though we generate a list of random words every time a user enrolls or tries to log in, there is a limited vocabulary that we are pulling from - namely English nouns provided in the RandomWords corpus - and words have been repeated in passphrases multiple times. For a malicious party, they could easily record a user saying these words, and, at some point, build up a large enough corpus such that they would be able to log in as the user simply by replaying the recorded audio spliced together (assuming they have large enough computing power to perform this task within a second). There are a couple of ways to fix this. First, we could expand our corpus to contain more words, which makes it harder for an attacker to record all of the words being spoken. Secondly, we could artificially introduce a noise into the signal recorded that is unique to that session, and therefore if the noise is detected in the recording, it will be discarded as a replay attack.

C. Voice Generation Attacks

Similar to replay attacks, we believe that there may be ways to regenerate an individual's voice in the future from a set number of samples. This would enable individuals to be able to synthesize audio from the challenge text that would be hard to discern for the system. While this is not currently an issue in the real world, we acknowledge that it could become an issue in the near future and it is a flaw that future iterations of the system should work to fix.

D. Securing Voice Print Data

As we mentioned earlier, while biometric information is unique, it's also something that cannot be changed easily. Thus, there is an even higher imperative to secure voice print data and the voice models. This would apply to all applications using voice biometric information, since if one application's samples were leaked, then other applications would be vulnerable to attacks as well.

VIII. Conclusion

We present an analysis of current voice authentication methods, a security policy as well as two different implementations of ways of supporting biometric two-factor authentication. Our implementations provide simple ways to be able to drop in two-factor authentication into existing applications, while also presenting ideas for future work. By presenting a voice biometric based authentication system, our system paves the way for additional work in open source biometric authentication systems. [19] [20]

References

[1] "bob.bio.spear 2.0.4–Tools for running speaker recognition experiments," https://pypi.python.org/pypi/bob.bio.spear/2.0.4

[2] J. Bonneau et. al., "The Quest To Replace Passwords," *IEEE Symposium on Security and Privacy*, 2012, pp. 553-567

[3] P. Cano et. al., "A Review of Audio Fingerprinting," *Journal of VLSI Signal Processing*, 2005, pp. 271-284

[4] "Dejavu: Audio Fingerprinting and Recognition in Python," https://github.com/worldveil/dejavu
[5] "Fernet (symmetric encryption)," https://cryptography.io/en/latest/fernet/
[6] "FuzzyWuzzy," https://github.com/seatgeek/fuzzywuzzy
[7] "Levenshtein Distance," https://en.wikipedia.org/wiki/Levenshtein_distance
[8] "Man-in-the-middle Attack," https://en.wikipedia.org/wiki/Man-in-the-middle_attack
[9] "Mel-Frequency Cepstrum," https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
[10] "Microsoft Cognitive Services: Speaker Recognition API," https://www.microsoft.com/cognitive-services/en-us/speaker-recognition-api
[11] L. Myers, "An Exploration of Voice Biometrics," 2004. https://www.sans.org/reading-room/whitepapers/authentication/exploration-voice-biometrics-1436
[12] "python_speech_features," https://github.com/jameslyons/python_speech_features
[13] J. Ramirez et al., "Efficient voice activity detection algorithms using long-term speech information," *Speech Communication*, 2004, pp. 271-287.
[14] "RandomWords 0.2.0–A Useful Module for Random Text, E-mails and Lorem Ipsum.," https://pypi.python.org/pypi/RandomWords
[15] D. Reynolds et al., "Speaker verification using adapted Gaussian Mixture models," *Digital Signal Processing*, 2000, pp. 19-41.
[16] "Simple-Minded Audio Classifier in Python," https://github.com/danstowell/smacpy
[17] "scikit-learn Documentation–sklearn.mixture.GMM," http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GMM.html
[18] "SpeechRecognition 3.4.3–Library for Performing Speech Recognition," https://pypi.python.org/pypi/SpeechRecognition/3.4.3
[19] "Why Fingerprints, Other Biometrics Don't Work," *USA Today*, http://www.usatoday.com/story/cybertruth/2013/09/12/why-biometrics-dont-work/2802095/
[20] "Why Haven't Biometrics Replaced Passwords Yet," *Digital Trends*, http://www.digitaltrends.com/android/can-biometrics-secure-our-digital-lives/
[21] X. Zhou et. al., "Digital Signal Processing: Speaker Recognition," https://raw.githubusercontent.com/ppwwyyxx/speaker-recognition/master/doc/Final-Report-Complete.pdf