

# Speaker Verification for Password Reset



A final year project submitted by

Matela Motlatsi 201903382 and Ramathebane Philemon 201800058

In partial fulfillment to the requirements for

B.Sc. Computer Science

To the

Department of Mathematics and Computer Science Faculty of Science and  
Technology

National University of Lesotho.

April 11, 2023

Supervisor: Dr. Lerato Lerato and Dr. Mathe Ntšekhe



# Acknowledgements

We would like to take this opportunity to extend our heartfelt thanks and gratitude to all those who have supported and assisted us throughout the course of this project - "Speaker Verification for Password Reset".

First and foremost, we would like to express our sincere appreciation and thanks to Mr. Lerato for his invaluable guidance, assistance and support throughout this project. We are deeply grateful for his generosity in sharing his knowledge, resources and expertise, particularly in relation to Asterisk, module.conf file, and his guidance and feedback on the project.

We would also like to thank Dr. Mathe for reviewing our project presentation slides and providing us with constructive feedback. His suggestions helped us to improve the quality of our work.

We also wish to thank our classmates for their contribution to the project, especially with data collection for enrollment in the system, offering critical feedback on the report, and assisting with some concepts. We would also like to express our gratitude to Mr. Serutla for his help in recording voice prompts for the dial plan on Telco.

Lastly, we would like to express our sincere gratitude and appreciation to our supervisors who provided feedback and guidance throughout the project, from the first presentation to the last. Their suggestions and insights have been instrumental in helping us implement the project successfully, along with developing an informative and comprehensive report. We would like to extend our deepest appreciation to everyone who contributed to the success of this project. Without their support, this endeavor would not have been possible. Thank you, all.



# Abstract

This paper summarizes the speaker verification for password reset project. The need for a safe, effective, and user-friendly solution for password reset operations is the main problem this project attempts to address. Agile is the technique used for this project; it is an incremental, iterative process that promotes adaptation and continual development. This technique has been essential to the project's ability to adapt and overcome obstacles.

The data used in this project and the test outcomes have several distinct elements. They include individual speakers' unique voice features, the accuracy of password resets made possible by speaker verification, and the overall user experience. The project's findings offer important new information on the possibility of speech verification for password reset. They illustrate how well this approach works to increase security while boosting user experience and also recommend areas that might require more development and study. In summary, this paper gives a thorough description of the project's problem statement, methodology, findings, and recommendations for speaker verification for password reset.

Copyright, 2023, by Motlatsi Matela and Philemon Ramathebane, All Rights Reserved.



## TABLE OF CONTENTS

	Page
Acknowledgements.....	ii
Abstract.....	iv
List of tables.....	xi
List of figures .....	xii
Chapter 1. Introduction.....	1
1.1 Background.....	1
1.2 Problem Statement .....	3
1.3 Requirements Lists .....	5
1.4 Use Case Diagram.....	6
1.5 Requirements Analysis .....	8
1.6 Scope of the project .....	8
1.7 Aim .....	9
1.8 Objectives.....	9
1.9 Tools and Technologies.....	10
1.10 Organization of the Document .....	12
Chapter 2. Literature Review.....	14
2.1 Password Reset approaches .....	14
2.1.1 Biometric Authentication approach.....	14
2.1.2 USSD Verification approach.....	15
2.1.3 Email Verification approach.....	17
2.1.4 SMS Verification approach.....	18
2.1.5 In-person Verification approach.....	18
2.1.6 Speaker Verification based password reset approach .....	19
2.1.6.1 Block diagram of Speaker recognition system .....	19
Chapter 3. System Design .....	21



3.1 Methodology .....	21
3.2 Functional and Non-Functional Requirements .....	23
3.2.1 Functional Requirements .....	23
3.2.2 Non-Functional Requirements .....	24
3.3 System Architecture .....	24
3.3.1 Full System Architecture .....	24
3.3.2 Speaker Verification .....	26
3.3.3 High Level Architecture .....	28
3.4 Hardware and Software Requirements .....	29
3.4.1 Hardware Requirements .....	29
3.4.2 Software Requirements .....	29
Chapter 4. Experiments and Results .....	30
4.1 Calling Platform Experiments .....	30
4.2. Speaker Verification Experiments .....	34
4.2.1 Stand-alone Speaker Verification Tests and Experiments .....	34
4.2.2 Stand-alone Speaker Verification Results using a known dataset .....	37
4.2.3 Using known Speaker Verification Evaluation Metrics such as False Rejection (FR) and False Acceptance (FA) .....	39
4.2.3.1 Experiment 1 .....	39
4.2.3.1.1 False Rejection .....	39
4.2.3.2 Experiment 2 .....	41
4.2.3.2.1 False Acceptance .....	41
4.2.3.3 Discussions on Using Speaker Verification Evaluation Metrics .....	42
4.2.3.3.1 Discussions on False Rejection .....	42
4.2.3.3.2 Discussions on False Acceptance .....	42
4.3 Complete System Performance .....	44
4.3.1 Test 1: Enrolling Caller into the System .....	44

4.3.1.1 Results and Discussions from the Experiment above .....	50
4.3.1.1.1 Results.....	50
4.3.1.1.2 Discussions.....	50
4.3.2 Test 2: Using System to Verify Caller .....	51
4.3.2.1 Results and Discussions from the Experiment above .....	58
4.3.2.1.1 Results.....	58
4.3.2.1.2 Discussions.....	58
Chapter 5. Conclusions and Future Work.....	59
5.1 Conclusions.....	59
5.1.1 Concluding on results from Enrollment.....	59
5.1.2 Concluding on results from Verification .....	59
5.2 Future work .....	60
Chapter 6. References.....	63
Chapter 7. Appendices.....	64
7.1 Appendix A .....	64
7.1.1 List of Abbreviations.....	64
7.2 Appendix B .....	65
7.2.1 System Manual .....	65
7.3 Appendix C.....	70
7.3.1 User Manual.....	70
7.4 Appendix D.....	71
7.4.1 Code Snippet .....	71
7.4.1.1 Sip.conf .....	71
7.4.1.2 Pjsip.conf.....	73
7.4.1.3 Extensions.conf .....	74
7.4.1.4 UBM_GMM.m.....	75
7.4.1.5 train_UBM.py.....	76

7.4.1.6 extensions.conf (enrollment).....	77
7.4.1.7 enrollment.py .....	78
7.4.1.8 enrollment.m.....	80
7.4.1.9 train_UserGMM.Py .....	81
7.4.1.10 mfcc.m.....	83
7.4.1.11 extensions.conf (verificaiton) .....	89
7.4.1.12 verification.py.....	91
7.4.1.13 verification.m .....	92
7.4.1.14 Verify_User.py .....	93
7.5 Appendix E .....	96
7.5.1 Project Schedule .....	96
7.5.2 Gantt Chart .....	97

## List of tables

	Page
Table 1 Incurred Costs.....	4
Table 2 Use Case and their functions .....	7
Table 3 Requirements Analysis.....	8
Table 4 Advantages and Disadvantages of Biometric Authentication approach .....	15
Table 5 Key Principles of Agile Methodology .....	21
Table 6 Key Practices of Agile Methodology .....	22
Table 7 Functional Requirements.....	23
Table 8 Non-Functional Requirements .....	24
Table 9 Hardware Requirements .....	29
Table 10 Software Requirements .....	29
Table 11 Issues essential but out of scope in the project .....	60
Table 12 Speaker Verification processes with their estimate time .....	61
Table 13 Additional features and functionalities .....	62

## List of figures

	Page
Figure 1 Use Case Diagram to Represent Requirements.....	6
Figure 2 Block diagram of Speaker Recognition System.....	19
Figure 3 Full System Architecture .....	25
Figure 4 Speaker Verification .....	27
Figure 5 High Level Architecture .....	28
Figure 6 SIP Phone.....	30
Figure 7 Dial-Plan .....	31
Figure 8 Using a known audio dataset from the TIMIT corpus.....	37
Figure 9 Enrolling the user into the system and training the model.....	37
Figure 10 Extracting the relevant features.....	38
Figure 11 Calculating log-likelihood scores using UBMGMM .....	38
Figure 12 Calculating log-likelihood scores using userGMM .....	38
Figure 13 Verifying the user .....	39
Figure 14 Enrolling users into the system.....	40
Figure 15 Verification successfully .....	40
Figure 16 14 Enrolling users into the system.....	41
Figure 17 Verification of users.....	42
Figure 18 Rejection of users .....	43
Figure 19 Running Asterisk.....	44
Figure 20 Sip-Phone .....	45
Figure 21 extensions.conf .....	46
Figure 22 sip.conf .....	46
Figure 23 pjsip.conf.....	46
Figure 24 Enrolment .....	47
Figure 25 Results from the Asterisk console .....	47
Figure 26 Capturing CallerID and current time .....	48
Figure 27 New directory created .....	48
Figure 28 Responses to the questions recorded .....	49
Figure 29 Storing audios files.....	49
Figure 30 Relevant features extracted from the recorded audio files.....	49

Figure 31 A caller specific GMM is trained .....	49
Figure 32 Informs the user they been successfully enrolled .....	50
Figure 33 Asterisk PBX running .....	51
Figure 34 Dialing 100 on a SIP Phone for verification .....	52
Figure 35 extensions.conf .....	53
Figure 36 sip.conf .....	53
Figure 37 pjsip.conf.....	53
Figure 38 Verification .....	54
Figure 39 Results from the Asterisk console .....	54
Figure 40 Capturing the CallerID and the current time .....	55
Figure 41 A new directory created.....	56
Figure 42 Responses from the question recorded.....	56
Figure 43 Storing audio files.....	56
Figure 44 Relevant features extracted from the audio files.....	56
Figure 45 Informing the user that is verified successfully .....	57
Figure 46 Showing that the caller is verified successfully .....	59
Figure 47 Project Schedule .....	96
Figure 48 Gantt Chart .....	97

# Chapter 1. Introduction

## 1.1 Background

Authentication systems are security measures put in place to secure data and systems by requiring additional input beyond a username and password for users to access a system. By providing this additional input, authentication systems help ensure that users are who they say they are. Authentication systems can require one other form of user input or more.

These systems are sometimes called MFA (Multiple-Factor Authentication). Most modern authentication systems offer a wider range of authentication methods than passwords alone. The most common approach for advanced authentication, such as two-factor authentication or multi-factor authentication, is to pair a password with some sort of external verification.

Users' identities are verified through authentication systems, which then enable them access to resources or services. For service access, there are several kinds of authentication methods, including:

1. Challenge questions.
2. Unique identifying items, such as physical devices or external applications.
3. Location-based authentication.
4. Password-based authentication: This sort of authentication system is the most common, requiring users to submit a username and password in order to access a service. By comparing the inputted password to a password that has already been saved, the system confirms the user's identity.
5. Two-factor authentication: The system requires users to provide two forms of authentication, such as a password and a one-time code sent to their mobile device. This adds an additional layer of security to the authentication process.

6. Biometric authentication (such as retinal and fingerprint scans or facial recognition): The system uses physical characteristics of the user, such as their fingerprint or facial recognition, to verify their identity.

The most popular technique for recovering lost passwords is via email confirmation. A user can ask for a password reset link to be sent to their registered email address if they forget their password. In order to reclaim access to their account, the user can then click the link and supply a new password. However, because hackers may access the user's email account and reset their password, this approach is susceptible to hacking and social engineering assaults.

Security question-based password recovery is an additional technique. To confirm their identity and change their password, users must respond to a series of pre-set questions, such as what color their favorite is or what their mother's maiden name is. However, because attackers may quickly get the user's personal information, this technique is also open to phishing threats.

Password recovery methods utilizing biometric authentication, such as speech verification, have gained popularity in recent years. As it is more challenging for attackers to get beyond biometric authentication, this approach can provide consumers with a more secure and convenient option to reset their passwords. Accuracy, dependability, and user-friendliness must all be carefully taken into account when implementing biometric authentication systems.

Using authentication improves data security and prevents potential breaches. When multi-factor authentication is required to access a system, the system is less vulnerable to security issues like weak passwords or attacks like phishing. Authentication systems are ideal for businesses with sensitive data or systems that require secure user accounts.<sup>[1]</sup>



## 1.2 Problem Statement

Services such as mobile money applications require authentication; a common one is the national identification document, which is to be presented to an agent to successfully register a user.

A recent trend seen is that mobile money operators have long queues at customer service centers for clients who have lost their PINs or passwords, and this also results in customers incurring costs of traveling to customer centers, and these costs are hefty as a result of the geographical nature of our country, and customers still have to present their national identification to be authenticated and enabled to reset their passwords.

The traffic at the call centers has also become too costly and time consuming for both clients and service providers. An alternative was proposed by the service providers, the use of Unstructured Supplementary Service Data (USSD) to reset a user's password.

But this alternative still poses some security threats, as family and friends can have access to a user's mobile device and can potentially initiate the password reset without the user's knowledge, and a person who has stolen a communication device can easily reset a password.

People who have access to the user's personal information, such as their email address or date of birth, can simply get around these measures. This could result in financial loss, identity theft, and illegal access to user accounts. For services like mobile money applications, authentication is required. Customers who have forgotten their PINs or passwords typically stand in long lines at mobile money operators.

Due to the fact that most password recovery procedures are manual, clients may also incur the following costs:

<b>Costs</b>	<b>Definition</b>
Time-consuming	Large lines at money services can be time-consuming since clients spend important time standing in line rather than finishing their transactions quickly.
Frustrating	Customers may have bad feelings as a result of waiting in long lines, which might harm their perception of the money service provider as a whole.
Security risks	Extended lines may make clients targets for theft or fraud as they wait in line, which poses a security issue.
Decreased productivity	Long lines can lower productivity for business customers who must perform financial transactions as part of their operations since they may have to wait for a lengthy time to finish their transactions.
Increased costs	Also, long lines can drive up expenses for money service providers since they may need to recruit more people to handle the lines or spend money on technology to speed up the process.

*Table 1 Incurred Costs*

Speaker recognition systems can be used as a means of remotely resetting the passwords. A traditional way of resetting a password is through USSD or email, which is not safe since a person who has stolen a communication device can easily reset a password when it comes to security.

Speaker verification is more secure than USSD since it uses biometric authentication, which is challenging to forge. Nevertheless, USSD is vulnerable to attacks like phishing and social engineering, in which attackers trick users into exposing their login information.

To solve this issue, the password reset procedure may be made more secure through the implementation of a speaker verification system. The system can prevent illegal access to user accounts and safeguard critical data by authenticating the user's identity through voice.

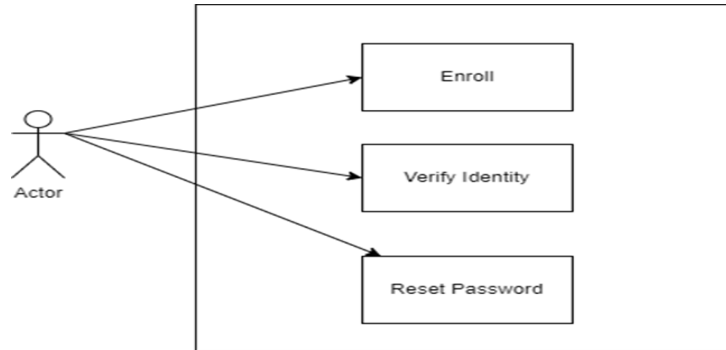
However, accuracy, dependability, user friendliness, and cost-effectiveness must all be carefully taken into account when designing and implementing a speaker verification system for password reset. The goal of this project is to create and implement a safe, dependable, user-friendly, and economical voice verification system for password reset.

## 1.3 Requirements Lists

1. Enrollment: Users should be capable of enrolling in the speaker verification system by repeatedly recording a mutiple files to produce a distinctive voiceprint.
2. Voiceprint storage: The system must keep the voiceprints safely in a database or other safe place.
3. Verification: The system should be able to recognize the user by their voiceprint and compare it to the voiceprint that has been previously saved in the database.
4. Voiceprint request: When a user requests a password reset, the system should ask for their voiceprint.
5. Prompt for new password: Only upon successful verification of the user's identification should the system then ask them to input a new password.
6. Password reset: After confirming the user's identification and asking them to input a new password, the system needs to be able to reset their password.
7. Security: The system should be created with security in mind, protecting the voiceprints and user data with safe storage, encryption, and access restrictions.
8. Accuracy: The system must be capable of accurately identifying the user and differentiating between voices and voiceprints.

9. Usability: During enrollment and verification, users should receive clear instructions and feedback on how to use and comprehend the system.
10. Integration: The system should be functional with established password reset and authentication systems as well as with established security frameworks and protocols.

## 1.4 Use Case Diagram



*Figure 1 Use Case Diagram to Represent Requirements*

Use Case	Function
Enroll	<p>The user speaks multiple phrases.</p> <p>The voice biometric system captures the voiceprint.</p> <p>Voiceprints are stored securely in the database.</p>
Verify Identity	<p>The user requests a password reset.</p> <p>The user is prompted to speak passphrase.</p> <p>The voice biometric system analyzes the voice and compares it to a stored voiceprint.</p> <p>The user identity is verified or rejected.</p>
Reset Password	<p>The user's identity is verified.</p> <p>The user is prompted to enter a new password.</p> <p>Password is reset.</p>

*Table 2 Use Case and their functions*

## 1.5 Requirements Analysis

Requirement	Definition
Entity-Relationship (ER) Diagram	Entities in the system's ER diagram might be Passphrase, User, Voiceprint, and Entity. The User entity would keep a user's name, email address, and password. The user's voiceprint would be stored in the Voiceprint object, while their recorded passphrase would be stored in the Passphrase entity. The user's password would be kept in the Password object and updated upon a successful password reset.
Database schema	Deduced from the ER diagram, the database schema may have tables with the relevant columns and relationships for User, Voiceprint, Passphrase, and Password.
User Interface	The user interface for the system could include a Sip-Phone for enrollment, verification, and password reset. The enrollment would prompt the user to record their passphrase multiple times to create a voiceprint, while the verification would prompt the user to provide their recorded passphrase for comparison to the stored voiceprint. The password reset would prompt the user to provide their verified identity and enter a new password.

*Table 3 Requirements Analysis*

## 1.6 Scope of the project

A speaker verification system for password resets has the purpose of adding an extra layer of protection to the password reset procedure by using the user's voice to confirm their identity. In order to verify the user's identity, this system would ask them to speak a passphrase or a list of predefined words or phrases, which would be matched to a voice sample that had already been recorded.

The system's weaknesses include the potential for false positives or false negatives, where it may mistakenly recognize a person as legitimate or reject a user who is actually authentic. The system's accuracy may also be impacted by outside variables, including

background noise, vocal changes brought on by sickness or exhaustion, or changes in the voice sample's recording quality.

A speaker verification system could also need extra hardware or software, which would raise the price and difficulty of the password reset procedure. Additionally, some users may not have access to a microphone or other recording equipment, which is another requirement of the system.

## 1.7 Aim

To implement a secure and reliable system that allows users to reset their passwords using their voice as authentication.

## 1.8 Objectives

1. Develop a speaker verification system that can accurately identify and authenticate users based on their voice.
2. Integrate the speaker verification system into the password reset process, allowing users to reset their passwords securely and easily.
3. To ensure that the speaker verification system is reliable and can accurately identify users even in noisy environments or when the user's voice changes due to illness or fatigue.
4. To ensure that the speaker verification system is user-friendly and easy to use, with clear instructions and prompts for users.

5. Then test the speaker verification system thoroughly to ensure that it meets the required security standards and is effective in preventing unauthorized access to user accounts.
6. Provide training and support to users and administrators to ensure that they can use the speaker verification system effectively and efficiently.
7. Then continuously monitor and improve the speaker verification system to ensure that it remains effective and up-to-date with the latest security standards and best practices.

## 1.9 Tools and Technologies

1. Asterisk PBX: this is an open-source software program that is used to enable a PC to run as a server for a VoIP service. This software will be used to switch calls between users on local lines. However, in this instance the calls will be switched between a caller and an automated service. Both enrollment and verification processes will be executed within the server, with all related audio recordings, features and models being stored within the server. Including the updated password.
2. Zoiper 5: this is a Session Initiation Protocol(SIP) phone, it manages multimedia communication such as video and voice calls. This will be used by a speaker, to initiate a call either for enrollment or verification.
3. Octave: is a high-level programming language primarily intended for scientific computing and numerical computation. It was used in this specific system to extract features from audio files, the extracted features are Mel-frequency Cepstral Coefficient. It is also used in managing the computation during the verification process.



4. Sci-kit learn: is a free software machine learning library for the Python programming language. This specific library contains the Gaussian Mixture Model(GMM) module, which when given an input of audio features, can train and fit a GMM to the features. It was used to train a Universal Background Model, including the user specific models.
5. Python; this is a general-purpose, interpreted, high-level programming language. This was used to build scripts that integrated all the above-mentioned technologies to perform both enrollment and verification.

## 1.10 Organization of the Document

Chapter 1. Introduction: Provides an overview of the project, its objectives, and the motivation behind it.

Chapter 2. Literature Review: Discusses relevant literature on remote password authentication systems, including password reset regimes, speaker verification, and voice authentication.

Chapter 3. System Design: Describes the methodology, functional and non-functional requirements, system architecture, hardware and software requirements, and implementation details of the project.

Chapter 4. Experiments and Results: Presents the results of experiments conducted on the Calling platform and speaker verification, and assesses the overall performance of the complete system.

Chapter 5. Conclusions and Future Work: Summarizes the findings of the project and outlines potential areas for future research and improvement.

Chapter 6. References: Cites all the sources used in the literature review and throughout the document.

Chapter 7. Appendices which Include the following:

List of abbreviations: Provides a list and explanation of any abbreviations and acronyms used in the document.

System manual: Provides technical instructions on the requirements, installation, and usage of the system, including details on hardware and software configuration and troubleshooting.

User manual: Describes how to use the implemented system, including step-by-step instructions for password reset using speaker verification.

Code: Includes the code used for implementing the system, organized into relevant sections and commented for clarity.

Project Schedule: A timeline or plan that outlines the tasks, activities, and milestones of a project that provides a roadmap for the project team to follow which helps in managing and tracking the project progress.

Gantt Chart: A visual representation of a project schedule illustrating start and end dates of each task, along with their durations.

## Chapter 2. Literature Review

### 2.1 Password Reset approaches

#### 2.1.1 Biometric Authentication approach

There is a type of biometric identification system called a pattern recognition system, where the system recognizes the user based on key features derived from specific behavioral characteristics (fingerprints, facial recognition, iris scans) of the person possessed for authentication purposes.

We have businesses such as E-Commerce that use this kind of system because they want to strengthen their platform to provide a safe shopping experience to the end consumer. That's why they have altered or defined new strategies with the passage of time. The ability to pay from the comfort of your home through credit cards or any other payment gateway has revolutionized the E-Commerce world.

To lower the risks associated with an online store, biometric technology has replaced conventional identity verification in favor of behavioral and physiological attributes. The most commonly used biometric is fingerprints, which needs the installation of a fingerprint reader and a database to capture and record the biometrics of users.

If a user forgot his or her password, the user will need to use his/her fingerprints to reset their password where the system reads fingerprints of a user and grants access for retrieving the password of the user only if it matches with the records that have been stored in the database, or else access is denied.

Advantages	Disadvantages
<p>Can be a fast and reliable method for resetting a password.</p> <p>Doesn't require the user to remember any specific information or answer security questions.</p>	<p>May not be available for all systems or services.</p> <p>Accuracy of biometric technology can vary depending on factors such as lighting conditions and the quality of the device's camera or sensor.</p> <p>Typically, it requires specialized hardware such as fingerprint scanners or facial recognition cameras. This can be expensive and may not be available on all devices.</p> <p>It can be less secure as biometric data can be stolen or spoofed, and there have been instances of biometric data breaches.</p>

*Table 4 Advantages and Disadvantages of Biometric Authentication approach*

### 2.1.2 USSD Verification approach

Some businesses could use USSD (Unstructured Supplementary Service Data) technology to allow consumers to reset their passwords. This technology enables businesses to connect with their clients using a menu-based system that is presented on their mobile phone screens. Businesses may offer services to their clients, such as password reset, over USSD in a simple and cost-effective manner.

Businesses can provide their clients a USSD code to call from their mobile phone so they can utilize USSD to change their passwords. Once the consumer dials the code, a menu allowing them to reset their password will be shown with some options to reset the password by either email, SMS, or by resolving security questions can be added to the menu.

Businesses can benefit from using USSD to change passwords in a number of ways. First off, because it doesn't need special devices or software, it is a cheap way to offer password reset services to consumers. Second, the system is straightforward and simple to use, requiring no technological expertise on the side of the user. Last but not least, USSD is a safe mechanism that may be used to verify the customer's identification prior to changing their password, guaranteeing that only authorized persons can access the account.

The quantity of information that can be presented and the kinds of interactions that may be made using USSD menus are limited. This might make it challenging to offer an adequate password reset procedure, particularly if extra verification procedures are necessary.

A mobile phone with a functional SIM card and network connection is necessary for USSD. Customers without access to a mobile phone or who are in a location with inadequate network coverage may find this to be an issue. Even though USSD has the potential to be a secure system, there is always the possibility of unwanted access or the collection of personal data. Businesses must implement the necessary safeguards, such as encryption and authentication mechanisms, to ensure the security of their USSD system.

USSD's dependency on the mobile network provider might cause problems with availability and availability. Customers might not be able to access the password reset service if the network is unavailable or having problems. USSD menus may not be as user-friendly as alternative methods of password reset and might be challenging to navigate. Customers may get frustrated and confused as a result, which may have a negative effect on how they see the business as a whole.

### 2.1.3 Email Verification approach

Additionally, multiple businesses offer email password reset services to their clients via emails. When a customer has forgotten his/her password, he/she will be required to enter his/her email address where the verification link will be sent in order to reset the password. If the provided email is wrong then the verification link will be sent to the wrong person which might lead to the person using the verification link to reset the password which does not belong to him/her.

So, the customer who is resetting the password has to be very careful when entering the email address to make sure that it is really correct. After the email is entered, there will be a verification link sent to the user's email address. The user then clicks the link to change their password and verify their identity.

Due to its simplicity and ease of use, this strategy is frequently utilized by enterprises. The verification link is delivered to the user's email address, which is normally a safe and private route, making it a secure one as well. However, this approach is susceptible to phishing attempts, in which criminals send fake verification links to a victim's email address in an effort to obtain their login information. This method's simplicity and accessibility are its advantages.

#### 2.1.4 SMS Verification approach

Users may retrieve their passwords with the use of SMS verification from businesses. This technique involves sending a verification code through SMS to the user's mobile device. The user then uses the code sent to reset their password and verify their identity; the user inputs the code.

Due to its simplicity and ease of usage, businesses also frequently use this technique. The verification code is delivered to the user's cell phone, which is normally a safe and private route, making it a secure way as well.

This technique, however, is disadvantageous in SIM swapping assaults, in which criminals take the victim's phone number and reroute the verification code.

#### 2.1.5 In-person Verification approach

This approach involves asking the user to reset their password and prove their identification in person at a physical place, such as a bank or retail outlet. It may be necessary for the user to travel with personal identification documents that will be needed to reset passwords once they arrive at their location. High-security applications can benefit from this technique, as can those without access to email or mobile devices.

Users who reside far from the real place or who have mobility challenges, however, may find this technique problematic. This approach has the benefit of offering a high level of security. Also, this approach may be disadvantageous where there might be long queues at the physical location where they will be assisted to reset their password.

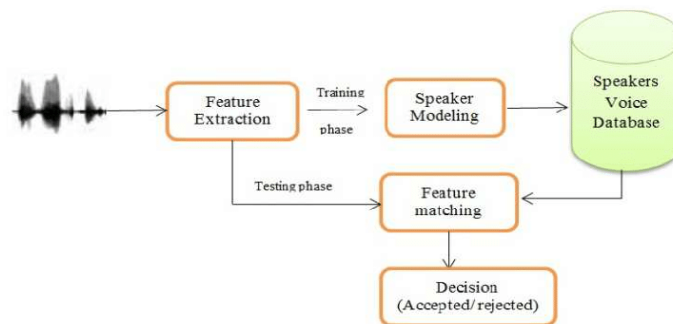
Again, imagine having customers who have traveled long distances from their homes find that there are long lines at the location, and unfortunately, they are not assisted, so they have to go back to their homes and come back in the next few days. This is very costly and time wasting.



### 2.1.6 Speaker Verification based password reset approach

Speaker verification is an identification technique that verifies an individual's identity by using the distinctive features of their speech. In this technique, the user's speech is recorded and analyzed, and a distinctive voiceprint is produced. The user's identification is verified by comparing the voiceprint to one that has been previously saved. The user is given access to the account or given the option to reset their password if their voiceprint matches. [2]

#### 2.1.6.1 Block diagram of Speaker recognition system



*Figure 2 Block diagram of Speaker Recognition System*

A range of technologies, including hardware- and software-based systems, can be used for speaker verification. Algorithms are used by software-based systems to analyze the user's voice and produce a voiceprint. Hardware-based systems record and analyze user voices using specialized hardware such as microphones and sound cards.

Compared to other authentication methods, speaker verification provides a number of benefits. As the user's voice is unique and difficult to imitate, it is a safe means of authentication. It is also a practical way because it can be carried out over the phone and needs no additional equipment or software. Speaker verification can also be used to

authenticate users who would have trouble doing so with normal techniques, such as people with impairments or those without access to a cell phone or email.

Here the speaker is used to capture the customer's voice, which will be used in training to know the unique voiceprints of each customer. When the customer now wants to reset their password, they will indeed use the speaker that will capture their voice, which will then be used to be matched against the voiceprint that has been stored in the database.

Speech biometrics are used in speech recognition as a kind of identification for verifying the user's identity. The user's identification is verified by comparing the voiceprint to one that has been previously saved. The user is given access to the account or given the option to reset their password if their voiceprint matches. Voice recognition is becoming more common since it offers a quick and safe way to authenticate. It may be used without any additional hardware or software, and it is simple to use. Voice recognition technology is already integrated into many products, giving users a simple choice.

Speech spoofing attacks, in which attackers use recordings of the user's speech to bypass authentication, can nonetheless make voice recognition vulnerable. If the voiceprint serves as the only method of authentication, this might pose a serious security issue.

From the above approaches, we have decided to implement speaker verification for password reset to assist customers in easily resetting their passwords.

## Chapter 3. System Design

### 3.1 Methodology

Agile software development methods allow for a flexible and iterative approach to development that can adapt to changing needs and user feedback, making them a suitable fit for speaker verification for password reset.

For this project, we have decided to use the Agile Project Methodology. Agile is an iterative and incremental project management methodology that emphasizes flexibility, collaboration, and customer satisfaction. It is an approach that can be applied to this specific project because it requires adaptability and frequent feedback.

The Agile project methodology is based on the following key principles<sup>[3]</sup>

Principle	Definition
Customer involvement	The customer will be actively involved in this project, providing feedback and direction throughout the development process. In the development of functionalities like training and testing of the model to be used, and finally testing whether the password change was successful through the use of a developed app.
Iterative development	The project has been broken down into smaller, more manageable parts that are developed in iterations or sprints. Each sprint delivers a working product increment that can be reviewed and adjusted based on feedback. Such parts include the development of the PBX server and the development of the speaker verification component, which is also broken down into training and testing.
Self-organizing teams	Individuals have been empowered to make decisions and manage their own workloads to achieve project goals.
Continuous improvement	The project team will be constantly looking for ways to improve the process and the product through regular feedback and retrospectives.
Flexibility and adaptability	The Agile methodology is designed to accommodate change and respond quickly to evolving requirements and customer needs.

*Table 5 Key Principles of Agile Methodology*

The Agile project methodology typically includes the following key practices<sup>[4]</sup>

Practice	Definition
Product backlog	A prioritized list of features and requirements that define speaker verification for the password reset project. The product backlog is constantly evolving and will guide the development team in planning and executing their work.
Sprint	In this phase, the team will identify the high priority items to be worked on, such as configuring the PBX server to receive calls from a certain extension and training and testing the speaker verification model to be able to correctly verify a user. The estimated times to complete these tasks will also be determined and followed. Sprint involves breaking down the product backlog items into smaller tasks and assigning them to the team members.
Daily stand-up	The team will hold a daily meeting to discuss the progress of each assigned task. Obstacles will then be identified and resolved. The plan for the day ahead will then be determined.
Sprint review	In this phase of the methodology, the team will demonstrate the functionality that has been developed during the sprint phase to the stakeholders. This is done for feedback and assessment.
Sprint Retrospective	After receiving feedback from stakeholders, the development team holds a retrospective meeting to discuss what went well, what did not go according to plan, and what can be improved in future sprints.

*Table 6 Key Practices of Agile Methodology*

These above-mentioned phases are repeated, and the team continues to work in sprints until all product backlog items have been implemented and tested to the satisfaction of the stakeholders.<sup>[5]</sup>

Agile is suitable for this project because it ensures that the product is developed in the planned period, continuously improved, and meets the needs of the users. Another benefit is that it allows for easy adaptation to changes in user requirements and any emerging issues.

This project report is a thorough manual for businesses looking to set up a dependable and secure speech verification system for password resets. Organizations may improve their security posture and give their consumers a more user-friendly experience by utilizing the potential of voice biometrics.

## 3.2 Functional and Non-Functional Requirements

### 3.2.1 Functional Requirements

Requirement	Definition
User authentication	The system should be able to authenticate the user's identity through their voice.
Password reset	Using voice verification, the system should make it simple and safe for users to change their passwords.
Voice recognition	The system must be able to accurately recognize the user's voice and then compare it with a voice sample that has already been recorded.
User interface	The system must have a user interface that is easy to use, leads users through the password reset procedure, and offers helpful feedback and clear instructions.

*Table 7 Functional Requirements*

### 3.2.2 Non-Functional Requirements

Requirement	Definition
Security	The system should be secure and protect user information from theft or unauthorized access.
Accuracy	When identifying the user's voice and verifying their identification, the system should be accurate and dependable.
Performance	The system should operate fast and effectively without problems or delays in the password reset procedure.
Usability	The system should be user-friendly and easy to use, with clear instructions and prompts for users.
Scalability	The system should be scalable and able to accommodate many users and requests.
Accessibility	Users with impairments, such as those who are blind or have restricted mobility, should be able to use the system.

*Table 8 Non-Functional Requirements*

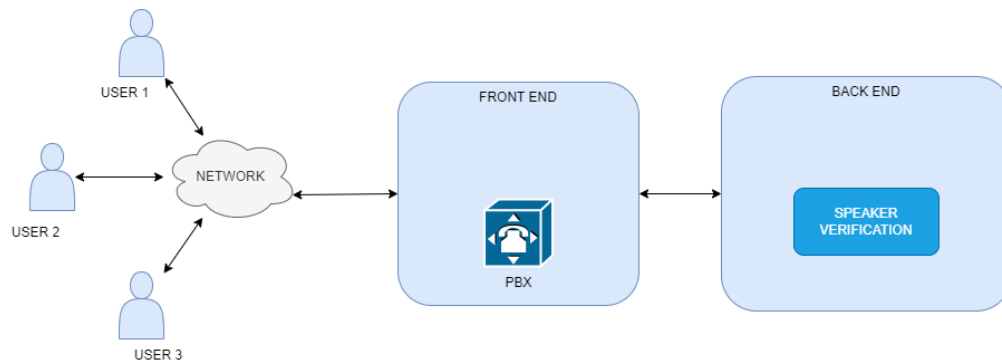
## 3.3 System Architecture

### 3.3.1 Full System Architecture

**Front-End:** This aspect of the system consists of the Private Exchange Branch, which is a telephone system in enterprises such as Vodacom or Econet Telecom Lesotho that offers multiple inbound and outbound calls and call routing. Voicemail and call management features.

This technology will be used to enrol users for the service. Audio will be recorded by the PBX and passed onto the speaker verification component. This is done so their features can be extracted. These extracted features will be used to create and train a voice model. The PBX system will also verify users using speaker verification to enable them to reset their password. This is done by recording the audio again, extracting features, and comparing those features to the voice models present in the system.

**Back-End:** This is the most vital aspect of the system, as it will be responsible for extracting features from recorded audios in the enrolment phase and using those features to create voice models. It will also be responsible for extracting features in the verification phase, using the same algorithm used in the enrolment instance, and comparing those features to voice models present in the system.



*Figure 3 Full System Architecture*

### 3.3.2 Speaker Verification

Here we have three phases been development, enrollment and verification phase. In development phase, we have a training dataset that is used to create the Universal Background Model (UBM) when we first extract features from all of the dataset then train the model using the extracted features thus saving the UBM which is a high order Gaussian Mixture Model (GMM) trained on a large quantity of speech. This phase is used to learn speaker-independent distribution of features, used in the alternative hypothesis in the likelihood ratio.

And in the enrollment phase, we can have two or more speakers to be enrolled in the system. They provide with their passphrase multiple times and be used to extract relevant features from the audios of the speakers then creating the user models of each speaker thus saving the user models.

Lastly, we have verification phase where we have a user claiming to reset their password and need to be verified before doing so. The claimer needs to provide with their passphrase (audio) which will then be used to extract relevant features from the provided audio then using the models (UBM and user models) created to verify the claimer. The score of the claimer identity GMM is computed in the enrollment phase. Then subtracting the score of the GMM of the UBM for each and obtaining the likelihood ratio. Therefore, comparing computed score to the threshold which will determine if the claimer if to be accepted or declined. <sup>[6]</sup>



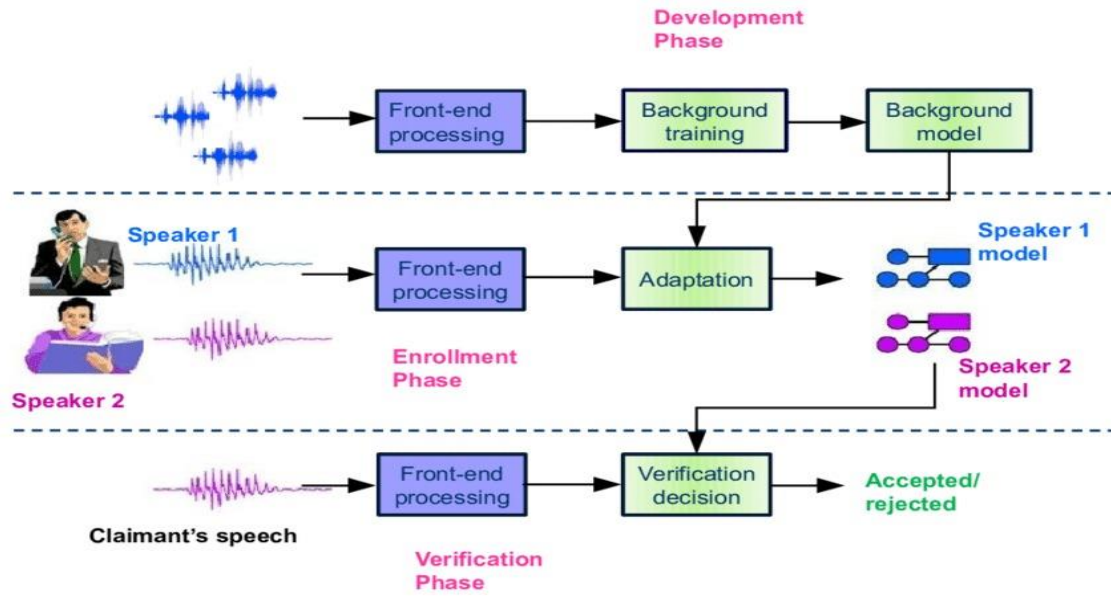


Figure 4 Speaker Verification

### 3.3.3 High Level Architecture

We have a Private Branch Exchange (PBX) which is a telephone system within the enterprise or organization that will accept calls from users. <sup>[7]</sup> When the user calls, he will be guided on what the user wants to do either for enrolment or for password change requests. Then if the user wants to be enrolled in the system, he will be requested to say his passphrase that will be automatically saved by firstly doing the feature extraction which are done by the speaker verification system then use the extracted features to train a voice model and store in the database.

If the user wants to reset their password, he will be requested to say his passphrase again where by the speaker verification system will do the same steps for enrolment to extract the features from the user's passphrase then use those extracted features to compare against the already stored models in the database. If the extracted features match with any of the models in the database, the user will be allowed to reset their password else not.

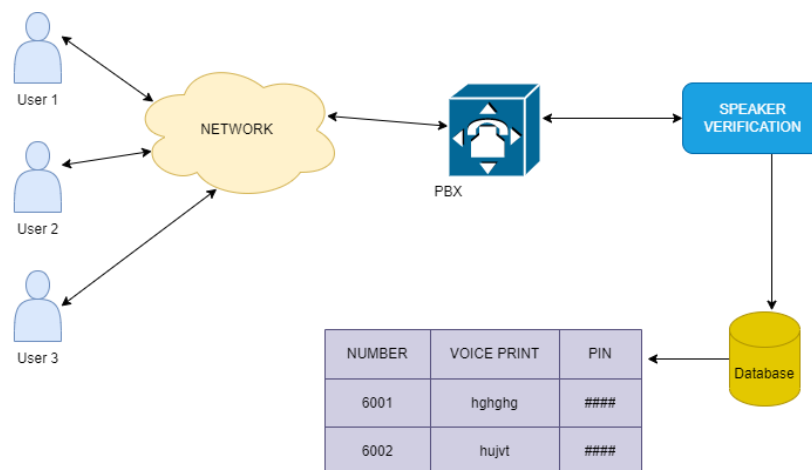


Figure 5 High Level Architecture

## 3.4 Hardware and Software Requirements

### 3.4.1 Hardware Requirements

Requirement	Definition
Microphone	A microphone is needed to record a speech sample for authentication and capture the user's voice.
Server	In order to save the speech samples and run the voice recognition algorithms, a server is needed (asterisk).
Network	In order to transfer the voice samples and authentication results between the user's device and the server, a network connection is necessary.

*Table 9 Hardware Requirements*

### 3.4.2 Software Requirements

Requirement	Definition
Voice recognition software	The system requires voice recognition software to accurately recognize the user's voice and compare it to a previously recorded voice sample.
Authentication software	The system requires authentication software to verify the user's identity based on their voice.
User interface software	The system requires user interface software to provide a user-friendly interface for the password reset process.
Database software	The system requires database software to store the voice samples and authentication results.

*Table 10 Software Requirements*

## Chapter 4. Experiments and Results

### 4.1 Calling Platform Experiments

Experiment: Using Asterisk PBX to Record User Speech

A SIP phone is used by the user to initiate a call, by dialing an extension on the SIP, in the experiment 101 was dialed, SIP phone used in this experiment was Zoiper. The call is transmitted to the Asterisk PBX, as it enables users to make calls on local lines.

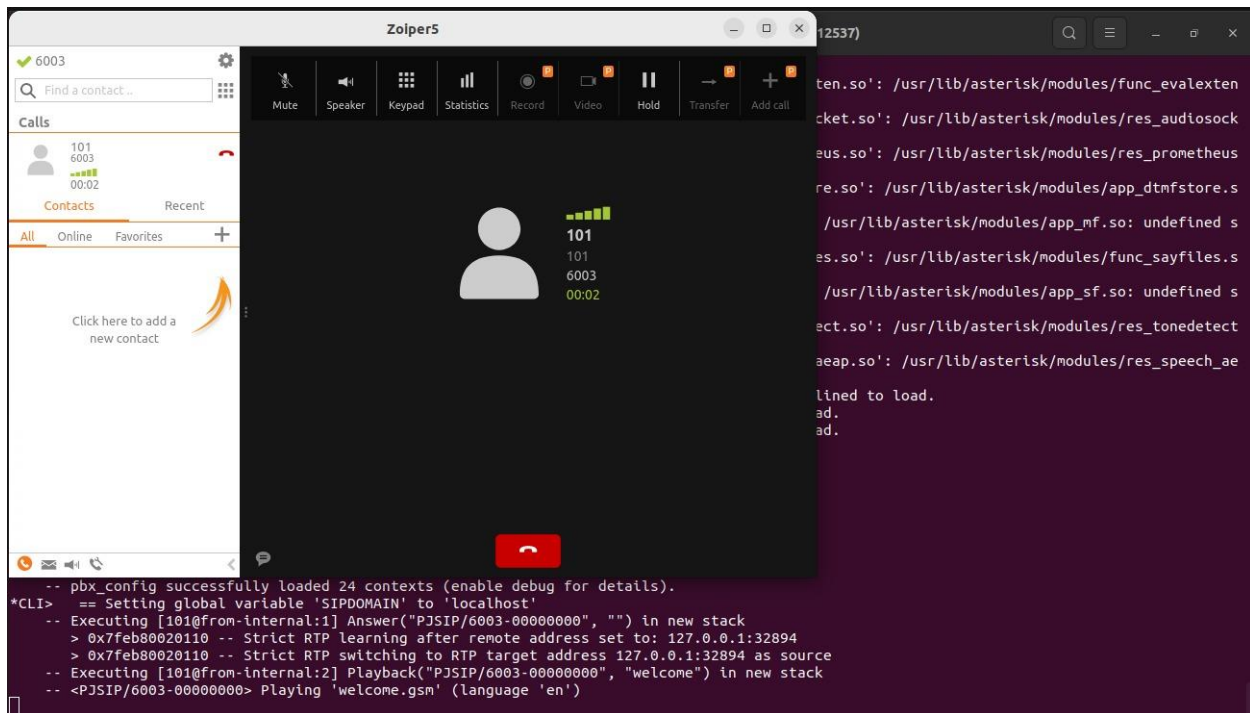


Figure 6 SIP Phone

The Asterisk PBX then initiates a dial-plan that is associated with the dialed extension, being 101. The dial-plan in a series of actions that will happen during the call. In this experiment, the following dial-plan was executed.

```

exten => 101,1,Answer()
same => n,Playback(welcome)
same => n,Set(CURRENT_TIME=${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
same => n,Set(CALLER_ID=${CALLERID(num)})
same => n,Set(i=1)

same => n,Playback(prompted_question)

; Create a new directory with the caller ID as the directory name
same => n,System(mkdir -p /home/setenane/Desktop/SV_for_Password_Reset/enrollment/${CALLER_ID})

; Loop to ask questions and record responses
same => n(loop),Playback(question_${i}) ; Playback the question message for the current iteration

; Set the recording duration based on the question number (i)
same => n,Set(RECORDING_DURATION=${IF(${i} == 1)?20:${IF(${i} == 2)?20:${IF(${i} == 3)?30:${IF(${i} == 4)?35:40}})}}))

same => n,Set(RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/UBM_training/${CALLER_ID}-${CURRENT_TIME}-${i}.wav)
same => n,Playback(beep) ; Play a beep sound before recording starts
same => n,Record(${RECORDED_FILE},${RECORDING_DURATION},10,16000)
same => n,Set(i=${i}+1)
same => n,GotoIf($[${i} < 6]?loop:end)
; Playback recorded successfully and thank you messages
same => n(end),Playback(thank_you)

; Hang up the call
same => n,Hangup()

```

*Figure 7 Dial-Plan*

## Explanation of the Code Snippet above

1. The code snippet above represents the dial-plan that will be executed when the extension 101 is dialed in the SIP phone dial-pad.
2. **exten => 101,1,Answer();** when the extension 101 is dialed, the call is answered by the Asterisk PBX.
3. **same => n,Playback(welcome);** the playback function is used to play an audio recording back to the caller, in this case a welcome message is played to the caller.
4. **same=>n,Set(CURRENT\_TIME=\${STRFTIME(\${EPOCH},,%Y%m%d-%H%M%S)});** this was used to capture the current time of the call.
5. **same => n,Set(CALLER\_ID=\${CALLERID(num)});** used to capture the callerID of the caller.
6. **same => n,Set(i=1);** the Set function used to initiate the value of i to 1.

7. **same => n,Playback(prompted\_question);** playback function plays an audio recording back to the caller, indicating that the caller will be prompted to answer a few questions.
8. **same=>n,System(mkdir-  
p/home/setenane/Desktop/SV\_for\_Password\_Reset/enrollment/\${CALLER\_ID})**  
; used to create a new directory, it will use the callerID as the directory name.
9. **same =>n(loop),Playback(question\_\${i});** This line will then initiate the loop, Playback the question message for the current iteration, the loop will be used to iterate through the different questions the caller will be asked during this session.
10. **same=>n,Set(RECORDING\_DURATION=\${IF(\${i}==1)?20:\${IF(\${i}==2)?20:\${IF(\${i}==3)?30:\${IF(\${i}==4)?35:40}}))});** used to set the recording duration based on which question is asked. Some questions require short answers, some may require longer responses.
11. **same=>n,Set(RECORDED\_FILE=/home/setenane/Desktop/SV\_for\_Password\_Reset/UBM\_training/\${CALLER\_ID}-\${CURRENT\_TIME}-\${i}.wav);** the location to which the audio will be stored is then initialized in this line, the recorded audio will be stored in the location home/setenane/Desktop/SV\_for\_Password\_Reset/UBM\_training. Each audio file will have a unique name, as the callerID will be used along with the timestamp of the recorded audio and an index, i, to count the number of audio files belonging to a single caller. The audio files will be stored as wav format.
12. **same=>n,Playback(beep);** this will playback a beep sound, this will indicate to the caller when the recording will start. Prompting the caller to utter words.

13. **same=>n,Record(\${RECORDED\_FILE},\${RECORDING\_DURATION},10,16000);**

this is where the Asterisk PBX then records the audio from the caller, using the location to where the audio will be stored after recording, the duration of the recording is also indicated, as they have the different durations. If the caller has not uttered a word within 10 seconds, the recording will be stopped, the sampling rate of the audio recording is then set to 16000Hz.

14. **same=>n,Set(i=\${i}+1));** Set function is used to increment the value of i, that was initialized earlier.

15. **same=>n,Gotof(\${i}<6?loop:end);** this Gotof function checks whether the value of i is less than 6, if not the loop will continue, and the next questions will be asked but once the variable i is equal to 6 or greater than 6, the loop is terminated.

16. **same=>n(end),Playback(thank\_you);** a thank you message is then played back to the caller.

17. **same=>n,Hangup();** the call is then hung up by the Asterisk PBX.

After the above dial-plan has finished executing, the audio files containing the caller's uttered speech have been recorded and stored in the indicated location.

## 4.2. Speaker Verification Experiments

### 4.2.1 Stand-alone Speaker Verification Tests and Experiments.

#### Step-by-step Description of How Speaker Verification Code Works

```
import numpy as np
import joblib
import sys

# Get the callerID argument passed from the command line
if len(sys.argv) < 2:
    print("Error: Missing callerID argument.")
    sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Load the saved features
savePath = f'/home/setenane/Desktop/SV_for_Password_Reset/Verification_features/{callerID}.dat'
allMFCCs = np.loadtxt(savePath)

# Load the universal background model
ubmModelPath = '/home/setenane/Desktop/SV_for_Password_Reset/Universal_Background_Model/GMM_UBM.pkl'
ubmGmm = joblib.load(ubmModelPath)

# Load the user-specific model
userModelPath = f'/home/setenane/Desktop/SV_for_Password_Reset/User_models/{callerID}.pkl'
userGmm = joblib.load(userModelPath)
```



```

# Perform speaker verification
ubmScores = ubmGmm.score_samples(allMFCCs.T)
userScores = userGmm.score_samples(allMFCCs.T)

# Set a threshold for verification
threshold = -10.0
verification = 0

# Compare the scores with the threshold
if userScores.mean() - ubmScores.mean() > threshold:
    verification = 1
else:
    verification = 0

if verification == 1:
    print(verification)
    print("Speaker has been Verified")
else:
    print(verification)
    print("Verification failed")

```

The above script will be used to execute the Verification component of the system.

1. At the beginning of the script, we import the necessary libraries and modules that will be used for data manipulation, model training, and saving/loading models.
  - a. First, import the NumPy library, this library allows the code to use its functions and data structures for numerical computations.
  - b. Secondly, import the joblib module, this provides the code utilities for saving and loading Python objects, for example models, either to or from the disk.

- c. Then, lastly, import the sys module, which provides access to some variables used or maintained by the interpreter and functions that interact with the interpreter.
2. The script then receives a parameter from the command line, this parameter is the callerID. If no argument is given, the code terminates before performing verification.
3. The callerID is then converted to a string value. This will allow the code to correctly find the directory that contains the caller's saved features.
4. The next step would then be to load the user's saved features, using the callerID to find the specific location in which they are stored in. These features have been extracted from the caller's recorded audio files. The features were extracted and stored when the caller attempted to be verified by the system.
5. The Universal Background Model is then loaded. This model contains users that have not been enrolled into the system. It is trained and fitted using extracted features from a large dataset containing audio files from different speakers, both males and females.
6. The caller's specific model is then loaded, this model has been trained and fitted using features extracted from the caller's recorded audio files. This model was trained when the caller was enrolled in the system.
7. The next lines of code perform an integral part of the verification process, using the Gaussian Mixture Models.
  - a. `ubmScores = ubmGmm.score_samples(allMFCCs.T)`: This line calculates the log-likelihood scores of the input MFCC features (`allMFCCs`) using the Universal Background Model (UBM) GMM (`ubmGmm`). The `score_samples` method returns the log-likelihood scores for each input sample.
  - b. `userScores = userGmm.score_samples(allMFCCs.T)`: This line calculates the log-likelihood scores of the input MFCC features (`allMFCCs`) using the user-specific GMM (`userGmm`). Similar to the previous line, the `score_samples` method returns the log-likelihood scores for each input sample.
8. These scores are then used to compare the user-specific model with the universal background model to determine if the speaker has been verified. The mean of the user scores is subtracted from the mean of the UBM scores, and if the difference is

greater than a predefined threshold, the verification is considered successful. Otherwise, the verification fails.

#### 4.2.2 Stand-alone Speaker Verification Results using a known dataset

Demonstrating through Results that Speaker Verification Works Using a Known Dataset, Speaker Recognition Corpus

1. Using a known audio dataset from the TIMIT corpus.

```
setenane@setenane-HP-250-G7-Notebook-PC:~$ cd Desktop/SV_for_Password_Reset/dataset/train/dr8/
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/dataset/train/dr8$ ls
fbcg1 fclt0 fklh0 fnkl0 mbcg0 mxcn0 mkdd0 mmea0 mmpm0 mrdm0 mrre0
fceg0 fjrb0 fmbg0 flpl0 mbsb0 mejs0 mkrge0 mmlm0 mmws0 mrlk0 mtcs0
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/dataset/train/dr8$ cd fbcg1/
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/dataset/train/dr8/fbcg1$ ls
sa1.phn sa2.phn si1612.phn si2242.phn si982.phn sx172.phn sx262.phn sx352.phn sx442.phn sx82.phn
sa1.txt sa2.txt si1612.txt si2242.txt si982.txt sx172.txt sx262.txt sx352.txt sx442.txt sx82.txt
sa1.wav sa2.wav si1612.wav si2242.wav si982.wav sx172.wav sx262.wav sx352.wav sx442.wav sx82.wav
sa1.wrd sa2.wrd si1612.wrd si2242.wrd si982.wrd sx172.wrd sx262.wrd sx352.wrd sx442.wrd sx82.wrd
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/dataset/train/dr8/fbcg1$
```

Figure 8 Using a known audio dataset from the TIMIT corpus

2. Before getting verified, using a portion of the audio dataset, enroll user into the system, by having relevant features extracted from the audio dataset. The features are then used to train a Gaussian Mixture Model.

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/enrollment/6008$ ls
sa1.wav sa2.wav si1612.wav si2242.wav si982.wav
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/enrollment/6008$

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6008
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6008
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$
```

Figure 9 Enrolling the user into the system and training the model

3. When getting verified, from the rest of the audio dataset, relevant features are extracted. Then the model trained during the enrollment phase is loaded along with the Universal Background Model.

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/verification/6008$ ls
sx172.wav  sx262.wav  sx352.wav  sx442.wav  sx82.wav
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/verification/6008$

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/nfcc$ python3 verification.py 6008
Feature extraction and saving completed.

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/nfcc$ python3 Verify_User.py 6008
Callers specific features loaded successfully
Universal Background Model loaded successfully
Callers specific Gaussian Mixture Model loaded successfully
```

*Figure 10 Extracting the relevant features*

4. Calculate the log-likelihood scores of the features using the Universal Background Model (UBM) GMM (ubmGmm). Then return the log-likelihood scores for each input sample.

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/nfcc$ python3 Verify_User.py 6008
Callers specific features loaded successfully
Universal Background Model loaded successfully
Callers specific Gaussian Mixture Model loaded successfully
ubmScores: [-30.23610933 -33.01066062 -33.50720258 ... -32.72275591 -34.66942566
-38.53624285]
```

*Figure 11 Calculating log-likelihood scores using UBMGMM*

5. Calculates the log-likelihood scores of the features using the user-specific GMM (userGmm). Similar to the previous line, returns the log-likelihood scores for each input sample.

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/nfcc$ python3 Verify_User.py 6008
Callers specific features loaded successfully
Universal Background Model loaded successfully
Callers specific Gaussian Mixture Model loaded successfully
ubmScores: [-30.23610933 -33.01066062 -33.50720258 ... -32.72275591 -34.66942566
-38.53624285]
userScores: [-31.59313286 -35.72843244 -35.5319906 ... -34.30449394 -36.06083006
-41.04528867]
```

*Figure 12 Calculating log-likelihood scores using userGMM*

6. These scores are then used to compare the user-specific model with the universal background model to determine if the speaker has been verified. The mean of the user scores is subtracted from the mean of the UBM scores, and if the difference is greater than a predefined threshold, the verification is considered successful. Otherwise, the verification fails.

```

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6008
Callers specific features loaded successfully
Universal Background Model loaded successfully
Callers specific Gaussian Mixture Model loaded successfully
ubmScores: [-30.23610933 -33.01066062 -33.50720258 ... -32.72275591 -34.66942566
-38.53624285]
userScores: [-31.59313286 -35.72843244 -35.5319906 ... -34.30449394 -36.06083006
-41.04528867]
Difference: -1.942168753348561
1
Speaker has been Verified

```

*Figure 13 Verifying the user*

### 4.2.3 Using known Speaker Verification Evaluation Metrics such as False Rejection (FR) and False Acceptance (FA)

#### 4.2.3.1 Experiment 1

##### 4.2.3.1.1 False Rejection

Set an experiment using 10 different speakers, 5 females and 5 males. Experiment set up to check whether the system will reject valid speakers. This is to check the performance of the speaker verification system.

- a. First step would be to enroll all speakers into the system. By extracting relevant features (MFCCs) then using those features to train speaker specific models.
- b. Secondly, if speaker has been enrolled successfully into the system. Verify their identity by extracting features from another set of audio files that belong to the same speaker, then compare the features to the already trained speaker specific model and the Universal Background Model.

The results from experiment are as follows:

1. All speakers were enrolled into the system successfully, the snapshot of the terminal below illustrates this:

```

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6010
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6010
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6011
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6011
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6012
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6012
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6013
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6013
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6014
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6014
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6015
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6015
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6016
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6016
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6017
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6017
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6018
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6018
Enrolled Successfully

```

Figure 14 Enrolling users into the system

2. All five female speakers were verified by the system, and all male speakers were verified by the system. The snapshot below illustrates this:

```

-0.6463930775954196
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6012
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6012
0.05518752203597188
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6013
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6013
-1.7219862823548482
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6014
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6014
-1.252625131702672
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6015
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6015
-1.3961799088550748
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6016
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6016
-2.0731808291140155
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6017
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6017
-0.8452588441954347
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6018
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6018
-1.3442050832533994
Speaker is Verified

```

Figure 15 Verification successfully

## 4.2.3.2 Experiment 2

### 4.2.3.2.1 False Acceptance

Set an experiment using 10 different speakers, 5 females and 5 males. Experiment set up to check whether the system will accept invalid speakers. This is to check the performance of the speaker verification system.

- First step would be to enroll all speakers into the system. By extracting relevant features (MFCCs) then using those features to train speaker specific models.
- Secondly, if speaker has been enrolled successfully into the system. Verify their identity by extracting features from another set of audio files that belong to another speaker, then compare the features to the already trained speaker specific model and the Universal Background Model.

The results from experiment are as follows:

- All speakers were enrolled into the system successfully, the snapshot of the terminal below illustrates this:

```
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6020
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6020
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6021
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6021
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6022
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6022
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6023
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6023
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6024
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6024
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6025
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6025
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6026
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6026
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6027
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6027
Enrolled Successfully
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 enrollment.py 6028
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 train_UserGMM.py 6028
Enrolled Successfully
```

Figure 16 Enrolling users into the system



2. The system has failed to reject all invalid speakers, it has verified speakers that have not been enrolled into the system. Snapshot below illustrates this:

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6022
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6022
-4.3823305514627435
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6023
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6023
-3.3333797677152575
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6024
Rhythmbox: Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6024
-4.296752536334871
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6025
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6025
-2.2913002319671847
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6026
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6026
-5.30460960035461
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6027
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6027
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6027
-3.028734820668703
Speaker is Verified
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 verification.py 6028
Feature extraction and saving completed.
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6028
-4.657416561751788
Speaker is Verified
```

Figure 17 Verification of users

#### 4.2.3.3 Discussions on Using Speaker Verification Evaluation Metrics

##### 4.2.3.3.1 Discussions on False Rejection

The results demonstrate that the speaker verification system has been able to verify all enrolled speakers. By verifying all valid speakers. This shows that there is a low chance that a valid speaker can be rejected by the system.

##### 4.2.3.3.2 Discussions on False Acceptance

The system has demonstrated to not be able to reject false speakers, by verifying speakers that have not been enrolled into the system.



A solution to the system falsely verifying users, is adjusting the threshold to a higher value, from threshold = -10.0 to threshold = -1. This can be seen demonstrated below, the same speakers that had been falsely verified by the system will be rejected:

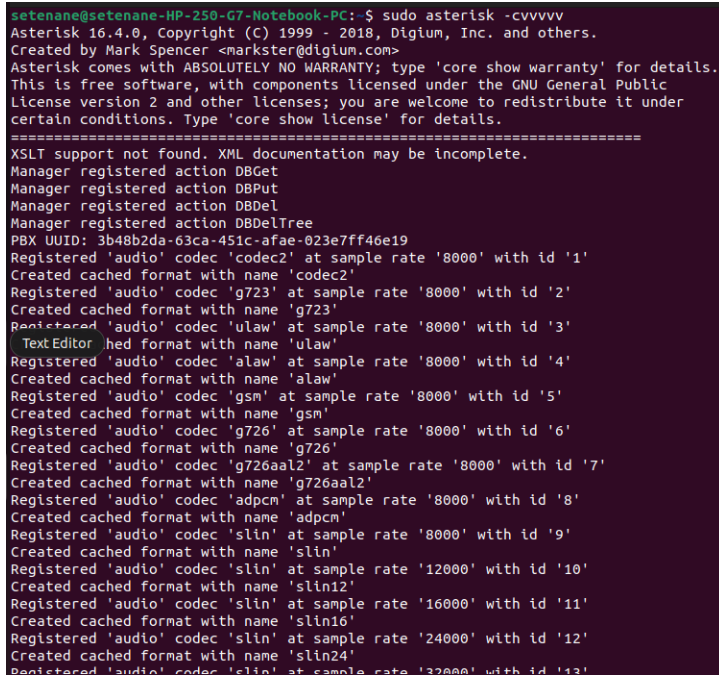
```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6019
-5.441161838931116
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6019
-5.441161838931116
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6020
-1.8503504257978207
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6021
-3.8433969979840157
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6022
-4.3823305514627435
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6023
-3.3333797677152575
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6024
-4.296752536334871
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6025
-2.2913002319671847
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6026
-5.30460960035461
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6027
-3.028734820668703
Verification failed
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc$ python3 Verify_User.py 6028
-4.657416561751788
Verification failed
```

*Figure 18 Rejection of users*

## 4.3 Complete System Performance

### 4.3.1 Test 1: Enrolling Caller into the System

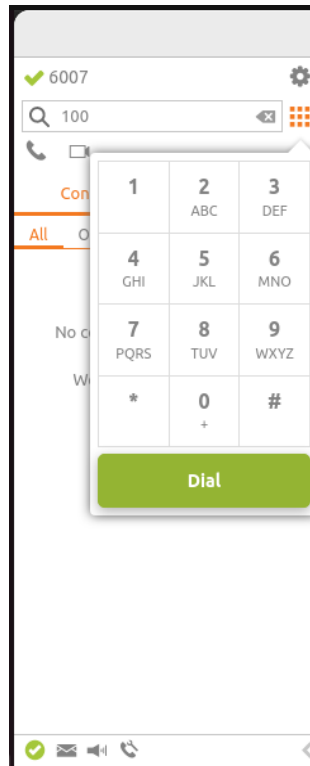
1. Before initiating a call, the Asterisk PBX should start running, this is accomplished by running `-sudo asterisk -cvvvvv` in the terminal.

A terminal window with a dark purple background and white text. The text shows the command 'sudo asterisk -cvvvvv' being executed. The output includes copyright information for Asterisk 16.4.0, a list of registered actions (DBGet, DBPut, DBDel, DBDelTree), the PBX UUID, and a series of messages registering various audio codecs (codecs) and creating cached formats for them. The codecs listed include 'codecs2', 'g723', 'ulaw', 'alaw', 'gsm', 'g726', 'g726aal2', 'adpcm', and 'slin' at different sample rates (8000, 12000, 16000, 24000, 32000 Hz).

```
setenane@setenane-HP-250-G7-Notebook-PC:~$ sudo asterisk -cvvvvv
Asterisk 16.4.0, Copyright (C) 1999 - 2018, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
XSLT support not found. XML documentation may be incomplete.
Manager registered action DBGet
Manager registered action DBPut
Manager registered action DBDel
Manager registered action DBDelTree
PBX UUID: 3b48b2da-63ca-451c-afae-023e7ff46e19
Registered 'audio' codec 'codecs2' at sample rate '8000' with id '1'
Created cached format with name 'codecs2'
Registered 'audio' codec 'g723' at sample rate '8000' with id '2'
Created cached format with name 'g723'
Registered 'audio' codec 'ulaw' at sample rate '8000' with id '3'
Created cached format with name 'ulaw'
Registered 'audio' codec 'alaw' at sample rate '8000' with id '4'
Created cached format with name 'alaw'
Registered 'audio' codec 'gsm' at sample rate '8000' with id '5'
Created cached format with name 'gsm'
Registered 'audio' codec 'g726' at sample rate '8000' with id '6'
Created cached format with name 'g726'
Registered 'audio' codec 'g726aal2' at sample rate '8000' with id '7'
Created cached format with name 'g726aal2'
Registered 'audio' codec 'adpcm' at sample rate '8000' with id '8'
Created cached format with name 'adpcm'
Registered 'audio' codec 'slin' at sample rate '8000' with id '9'
Created cached format with name 'slin'
Registered 'audio' codec 'slin' at sample rate '12000' with id '10'
Created cached format with name 'slin12'
Registered 'audio' codec 'slin' at sample rate '16000' with id '11'
Created cached format with name 'slin16'
Registered 'audio' codec 'slin' at sample rate '24000' with id '12'
Created cached format with name 'slin24'
Registered 'audio' codec 'slin' at sample rate '32000' with id '13'
```

*Figure 19 Running Asterisk*

2. When enrolling, a caller dials the extension 100 in a SIP phone. In this experiment, the SIP phone used is Zoiper 5.



*Figure 20 Sip-Phone*

3. Since the Asterisk PBX is already running, the call placed in the SIP phone will be answered since the extension is defined in the Asterisk extensions.conf module and the callerID has been configured into the Asterisk PBX.

```
GNU nano 6.2                                extensions.conf *
[from-internal]
exten => t,1,NoOp()
exten => e,1,NoOp()

exten => 100,1,Answer()
same => n,Playback(welcome)
same => n,Playback(press_1)
same => n,Playback(press_2)
same => n,WaitExten()

exten => 1,1,Goto(select-option,1,1)
exten => 2,1,Goto(select-option,2,1)
```

Figure 21 extensions.conf

```
GNU nano 6.2                                sip.conf *
[general]
context=default

[6007]
type=friend
context=from-internal
host=dynamic
secret=1234
;disallow=all
allow=all
```

Figure 22 sip.conf

```
GNU nano 6.2                                pjsip.conf *
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[6007]
type=endpoint
context=from-internal
;disallow=all
allow=all
auth=6007
aors=6007
;expiration=3600 ; Set expiration time to 3600 seconds (1 hour)

[6007]
type=auth
auth_type=userpass
password=1234
username=6007

[6007]
type=aor
max_contacts=1
```

Figure 23 pjsip.conf

- After the call has been answered, a welcome message is played back to the caller. The caller is then informed of the options available to them, since the caller is enrolling, the caller will enter 1 into the dial pad. The dial-plan will receive an input from the SIP phone, since the input is 1, the flow of execution of execution in the dial-plan will follow accordingly.

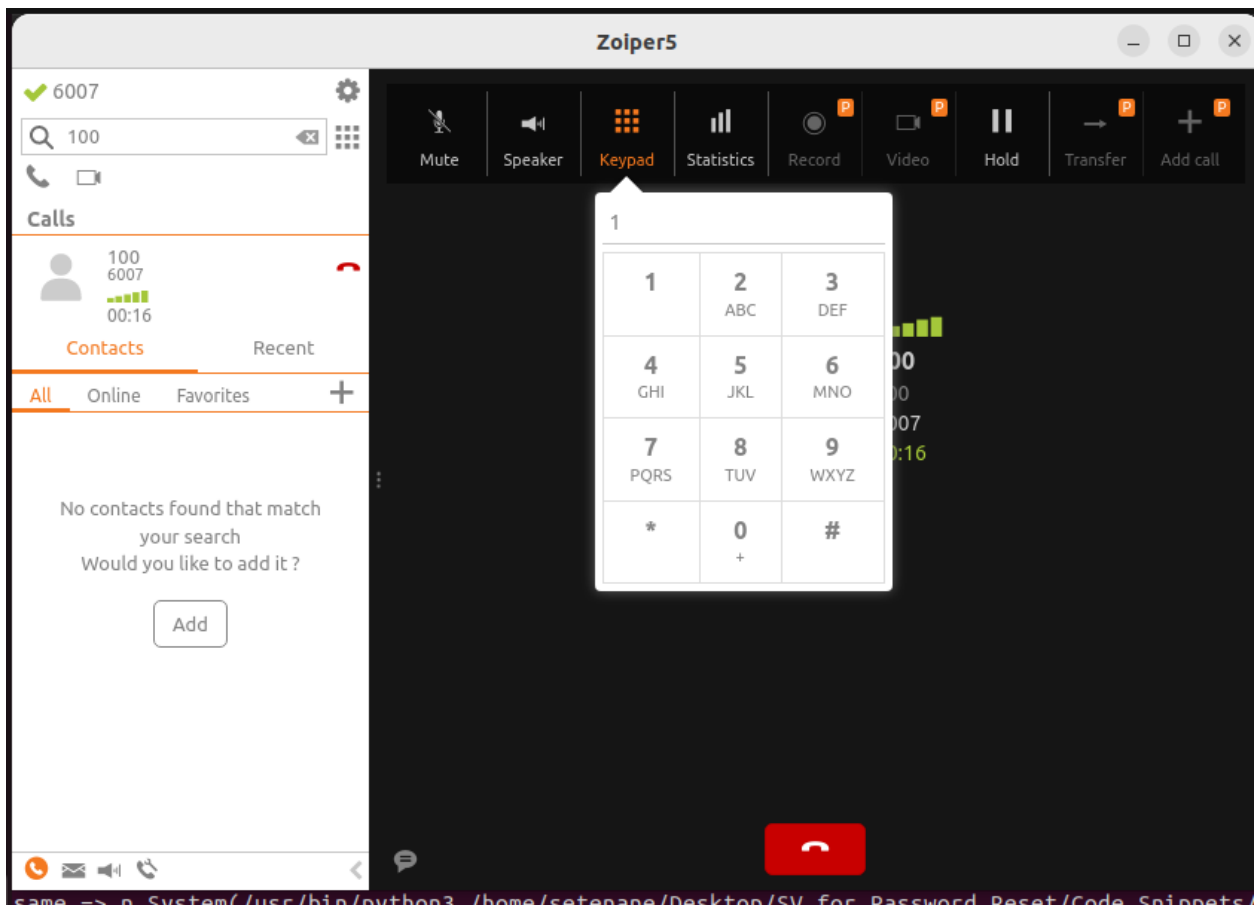


Figure 24 Enrolment

Below is results from the Asterisk console;

```
-- Executing [100@from-internal:1] Answer("PJSIP/6007-00000000", "") in new stack
> 0x7f243c0354b0 -- Strict RTP learning after remote address set to: 127.0.0.1:42581
-- Executing [100@from-internal:2] Playback("PJSIP/6007-00000000", "welcome") in new stack
-- <PJSIP/6007-00000000> Playing 'welcome.gsm' (language 'en')
> 0x7f243c0354b0 -- Strict RTP switching to RTP target address 127.0.0.1:42581 as source
> 0x7f243c0354b0 -- Strict RTP learning complete - Locking on source address 127.0.0.1:42581
-- Executing [100@from-internal:3] Playback("PJSIP/6007-00000000", "press_1") in new stack
-- <PJSIP/6007-00000000> Playing 'press_1.gsm' (language 'en')
-- Executing [100@from-internal:4] Playback("PJSIP/6007-00000000", "press_2") in new stack
-- <PJSIP/6007-00000000> Playing 'press_2.gsm' (language 'en')
-- Executing [100@from-internal:5] WaitExten("PJSIP/6007-00000000", "") in new stack
-- Executing [10@from-internal:1] Goto("PJSIP/6007-00000000", "select-option,1,1") in new stack
-- Goto (select-option,1,1)
```

Figure 25 Results from the Asterisk console

5. Since the input is 1, the flow of execution will lead to the following part of the dial-plan. In this dial-plan, the callerID will be captured and stored in a variable, so will the current time. These will be used to name the individual audio files that will be recorded.

```

-- Executing [100@from-internal:1] Answer("PJSIP/6007-00000000", "") in new stack
> 0x7f243c0354b0 -- Strict RTP learning after remote address set to: 127.0.0.1:42581
-- Executing [100@from-internal:2] Playback("PJSIP/6007-00000000", "welcome") in new stack
-- <PJSIP/6007-00000000> Playing 'welcome.gsm' (language 'en')
> 0x7f243c0354b0 -- Strict RTP switching to RTP target address 127.0.0.1:42581 as source
> 0x7f243c0354b0 -- Strict RTP learning complete - Locking on source address 127.0.0.1:42581
-- Executing [100@from-internal:3] Playback("PJSIP/6007-00000000", "press_1") in new stack
-- <PJSIP/6007-00000000> Playing 'press_1.gsm' (language 'en')
-- Executing [100@from-internal:4] Playback("PJSIP/6007-00000000", "press_2") in new stack
-- <PJSIP/6007-00000000> Playing 'press_2.gsm' (language 'en')
-- Executing [100@from-internal:5] WaitExten("PJSIP/6007-00000000", "") in new stack
-- Executing [1@from-internal:1] Goto("PJSIP/6007-00000000", "select-option,1,1") in new stack
-- Goto (select-option,1,1)
-- Executing [1@select-option:1] Answer("PJSIP/6007-00000000", "") in new stack
-- Executing [1@select-option:2] Playback("PJSIP/6007-00000000", "select_1") in new stack
-- <PJSIP/6007-00000000> Playing 'select_1.gsm' (language 'en')
-- Executing [1@select-option:3] Set("PJSIP/6007-00000000", "CURRENT_TIME=20230823-144743") in new stack
-- Executing [1@select-option:4] Set("PJSIP/6007-00000000", "CALLER_ID=6007") in new stack
-- Executing [1@select-option:5] Set("PJSIP/6007-00000000", "i=1") in new stack
-- Executing [1@select-option:6] Playback("PJSIP/6007-00000000", "prompted_question") in new stack
-- <PJSIP/6007-00000000> Playing 'prompted_question.gsm' (language 'en')

```

```

GNU nano 6.2 extensions.conf
[select-option]
exten => 1,1,Answer()
same => n,Playback(select_1)
same => n,Set(CURRENT_TIME=${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
same => n,Set(CALLER_ID=${CALLERID(num)})
same => n,Set(i=1)

same => n,Playback(prompted_question)

; Create a new directory with the caller ID as the directory name
same => n,System(mkdir -p /home/setenane/Desktop/SV_for_Password_Reset/enrollment/${CALLER_ID})

; Loop to ask questions and record responses
same => n(loop),Playback(question_${i}) ; Playback the question message for the current iteration

; Set the recording duration based on the question number (i)
same => n,Set(RECORDING_DURATION=${IF($[${i} == 1]?20:${IF($[${i} == 2]?20:${IF($[${i} == 3]?30:${IF($[${i} == 4]?35:40)}}))})

same => n,Set(RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/enrollment/${CALLER_ID}/${CALLER_ID}-${CURRENT_TIME}-${i}.wav)
same => n,Playback(beep) ; Play a beep sound before recording starts
same => n,Record(${RECORDED_FILE},${RECORDING_DURATION},10,16000)
same => n,Set(i=${i}+1)
same => n,GotoIf($[${i} < 6]?loop:end)

same => n(end),Playback(please_wait_processing)

; Execute the enrollment.py script using a system call, passing the callerID as an argument
same => n,System(/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/enrollment.py ${CALLER_ID})

same => n,System(/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/train_UserGMM.py ${CALLER_ID})

; Handle enrollment success
same => n,Playback(enrolled_successfully)

```

Figure 26 Capturing CallerID and current time

6. During the execution of the dial-plan, a new directory will be created, this directory will be used to store the caller's audio files.

```

setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/enrollment/6007$ ls
6007-20230823-145443-1.wav 6007-20230823-145443-2.wav 6007-20230823-145443-3.wav 6007-20230823-145443-4.wav 6007-20230823-145443-5.wav
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/enrollment/6007$

```

Figure 27 New directory created

7. A loop is used to iterate over 5 questions that have been pre-recorded, the caller will be prompted to provide answers to these questions and the responses will be recorded.

```
-- Executing [1@select-option:5] Set("PJSIP/6007-00000001", "i=1") in new stack
-- Executing [1@select-option:6] Playback("PJSIP/6007-00000001", "prompted_question") in new stack
-- <PJSIP/6007-00000001> Playing 'prompted_question.gsm' (language 'en')
-- Executing [1@select-option:7] System("PJSIP/6007-00000001", "mkdir -p /home/setenane/Desktop/SV_for_Password_Reset/enrollment/6007") in
new stack
-- Executing [1@select-option:8] Playback("PJSIP/6007-00000001", "question_1") in new stack
-- <PJSIP/6007-00000001> Playing 'question_1.gsm' (language 'en')
-- Executing [1@select-option:9] Set("PJSIP/6007-00000001", "RECORDING_DURATION=20") in new stack
-- Executing [1@select-option:10] Set("PJSIP/6007-00000001", "RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/enrollment/6007/6
007-20230823-145443-1.wav") in new stack
-- Executing [1@select-option:11] Playback("PJSIP/6007-00000001", "beep") in new stack
```

*Figure 28 Responses to the questions recorded*

8. After recording, the audio files will be stored in the created directory. Using the callerID, current time and an index,i.

```
-- Executing [1@select-option:11] Playback("PJSIP/6007-00000001", "beep") in new stack
-- <PJSIP/6007-00000001> Playing 'beep.gsm' (language 'en')
-- Executing [1@select-option:12] Record("PJSIP/6007-00000001", "/home/setenane/Desktop/SV_for_Password_Reset/enrollment/6007/6007-2023082
3-145443-5.wav,40,10,16000") in new stack
```

*Figure 29 Storing audios files*

9. The caller will then be informed that there is some processing happening, while the caller waits, the dial-plan will execute two scripts. Firstly, enrollment.py will be executed, the dial-plan passes the callerID to the script, this script will load the recorded audio from the created directory, which uses callerID as its name. Then from the recorded audio files, relevant features are extracted, MFCCs.

```
-- <PJSIP/6007-00000001> Playing 'please_wait_processing.gsm' (language 'en')
-- Executing [1@select-option:16] System("PJSIP/6007-00000001", "/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippe
ts/mfcc/enrollment.py 6007") in new stack
Feature extraction and saving completed.
```

*Figure 30 Relevant features extracted from the recorded audio files*

10. The second script to be executed is, train\_USerGMM, again the callerID is given to the script as an argument, the script will load the extracted features using the callerID, to locate features specific to the caller. Using the loaded features, a caller specific Gaussian Mixture Model is trained.

```
-- Executing [1@select-option:17] System("PJSIP/6007-00000001", "/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippe
ts/mfcc/train_UserGMM.py 6007") in new stack
Enrolled Successfully
```

*Figure 31 A caller specific GMM is trained*

11. Then the caller is informed that they have been enrolled successfully.

```
-- Executing [1@select-option:18] Playback("PJSIP/6007-00000001", "enrolled_successfully") in new stack
-- <PJSIP/6007-00000001> Playing 'enrolled_successfully.gsm' (language 'en')
-- Executing [1@select-option:19] Playback("PJSIP/6007-00000001", "thank_you") in new stack
-- <PJSIP/6007-00000001> Playing 'thank_you.gsm' (language 'en')
-- Executing [1@select-option:20] Hangup("PJSIP/6007-00000001", "") in new stack
```

*Figure 32 Informs the user they been successfully enrolled*

#### 4.3.1.1 Results and Discussions from the Experiment above

##### 4.3.1.1.1 Results

- a. When a caller placed a call, the call was answered,
- b. The caller then gives input to the dial-plan to indicate the caller wants to enroll for the service.
- c. The callerID was captured and used to create a directory that will store the caller's recorded audio files.
- d. The caller is asked questions and the caller's responses are stored in the specified directory.
- e. The script enrollment.py executed in the dial-plan, this script was able to load the audio files from their stored location, and extract relevant features (MFCCs) from the caller's audio files. The extracted features are then stored appropriately.
- f. The script train\_UserGMM.py executed in the dial-plan is able to load the features stored and use them to train a Gaussian Mixture Model. The trained model is then stored appropriately.

##### 4.3.1.1.2 Discussions

Using the different technologies and tools, we were able to achieve enrolling a caller into the system. By recording their audio files and storing them, extracting the necessary features from the audio recordings then using those features to train a Gaussian Mixture Model.



### 4.3.2 Test 2: Using System to Verify Caller

1. Before initiating a call, the Asterisk PBX should start running, this is accomplished by running `-sudo asterisk -cvvvvv` in the terminal.

```
setenane@setenane-HP-250-G7-Notebook-PC: $ sudo asterisk -cvvvvv
Asterisk 16.4.0, Copyright (C) 1999 - 2018, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
XSLT support not found. XML documentation may be incomplete.
Manager registered action DBGet
Manager registered action DBPut
Manager registered action DBDel
Manager registered action DBDelTree
PBX UUID: 3b48b2da-63ca-451c-afae-023e7ff46e19
Registered 'audio' codec 'code2' at sample rate '8000' with id '1'
Created cached format with name 'code2'
Registered 'audio' codec 'g723' at sample rate '8000' with id '2'
Created cached format with name 'g723'
Registered 'audio' codec 'ulaw' at sample rate '8000' with id '3'
  TextEditor  hed format with name 'ulaw'
Registered 'audio' codec 'alaw' at sample rate '8000' with id '4'
Created cached format with name 'alaw'
Registered 'audio' codec 'gsm' at sample rate '8000' with id '5'
Created cached format with name 'gsm'
Registered 'audio' codec 'g726' at sample rate '8000' with id '6'
Created cached format with name 'g726'
Registered 'audio' codec 'g726aal2' at sample rate '8000' with id '7'
Created cached format with name 'g726aal2'
Registered 'audio' codec 'adpcm' at sample rate '8000' with id '8'
Created cached format with name 'adpcm'
Registered 'audio' codec 'slin' at sample rate '8000' with id '9'
Created cached format with name 'slin'
Registered 'audio' codec 'slin' at sample rate '12000' with id '10'
Created cached format with name 'slin12'
Registered 'audio' codec 'slin' at sample rate '16000' with id '11'
Created cached format with name 'slin16'
Registered 'audio' codec 'slin' at sample rate '24000' with id '12'
Created cached format with name 'slin24'
Registered 'audio' codec 'slin' at sample rate '32000' with id '13'
```

*Figure 33 Asterisk PBX running*

2. To get verified, a caller dials the extension 100 in a SIP phone. In this experiment, the SIP phone used is Zoiper 5.

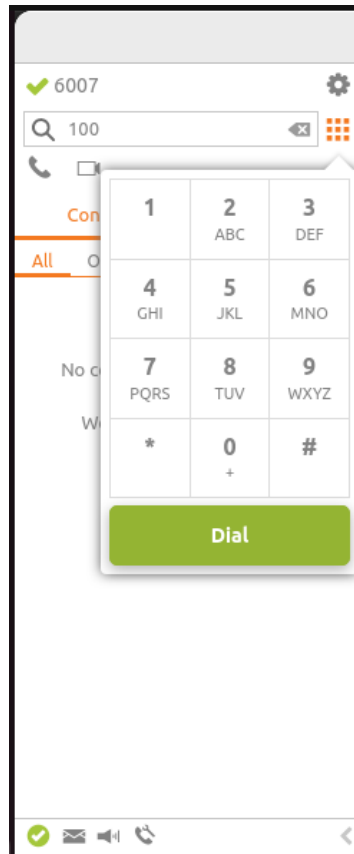


Figure 34 Dialing 100 on a SIP Phone for verification

3. Since the Asterisk PBX is already running, the call placed in the SIP phone will be answered since the extension is defined in the Asterisk extensions.conf module and the callerID has been configured into the Asterisk PBX.

```
GNU nano 6.2 extensions.conf *
[from-internal]
exten => t,1,NoOp()
exten => e,1,NoOp()

exten => 100,1,Answer()
same => n,Playback(welcome)
same => n,Playback(press_1)
same => n,Playback(press_2)
same => n,WaitExten()

exten => 1,1,Goto(select-option,1,1)
exten => 2,1,Goto(select-option,2,1)
```

Figure 35 extensions.conf

```
GNU nano 6.2 sip.conf *
[general]
context=default

[6007]
type=friend
context=from-internal
host=dynamic
secret=1234
;disallow=all
allow=all
```

Figure 36 sip.conf

```
GNU nano 6.2 pjsip.conf *
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[6007]
type=endpoint
context=from-internal
;disallow=all
allow=all
auth=6007
aors=6007
;expiration=3600 ; Set expiration time to 3600 seconds (1 hour)

[6007]
type=auth
auth_type=userpass
password=1234
username=6007

[6007]
type=aor
max_contacts=1
```

Figure 37 pjsip.conf

4. After the call has been answered, a welcome message is played back to the caller. The caller is then informed of the options available to them, since the caller is getting verified, the caller will enter 2 into the dial pad. The dial-plan will receive an input from the SIP phone, since the input is 2, the flow of execution of execution in the dial-plan will follow accordingly.

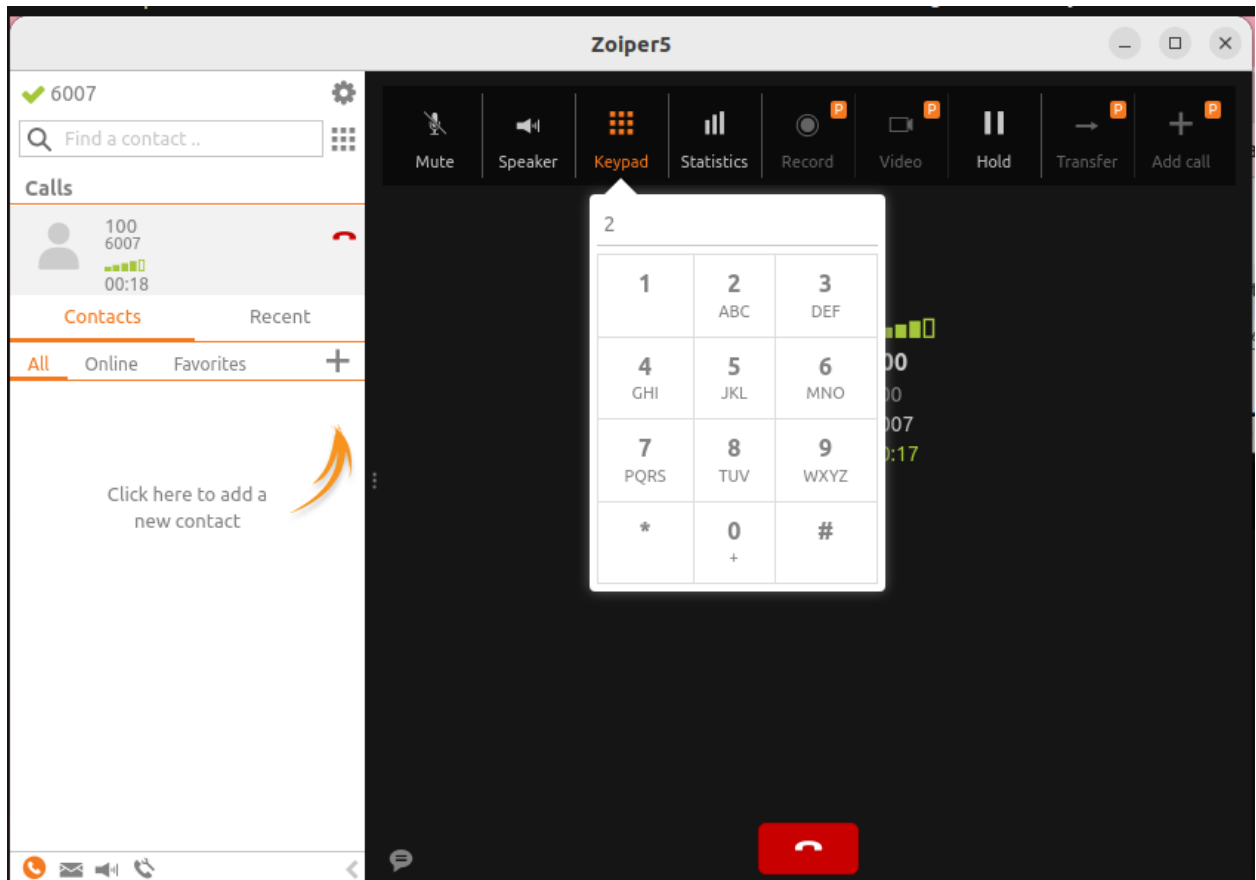


Figure 38 Verification

Below is results from the Asterisk console;

```
-- Executing [100@from-internal:1] Answer("PJSIP/6007-00000000", "") in new stack
> 0x7f243c0354b0 -- Strict RTP learning after remote address set to: 127.0.0.1:42581
-- Executing [100@from-internal:2] Playback("PJSIP/6007-00000000", "welcome") in new stack
-- <PJSIP/6007-00000000> Playing 'welcome.gsm' (language 'en')
> 0x7f243c0354b0 -- Strict RTP switching to RTP target address 127.0.0.1:42581 as source
> 0x7f243c0354b0 -- Strict RTP learning complete - Locking on source address 127.0.0.1:42581
-- Executing [100@from-internal:3] Playback("PJSIP/6007-00000000", "press_1") in new stack
-- <PJSIP/6007-00000000> Playing 'press_1.gsm' (language 'en')
-- Executing [100@from-internal:4] Playback("PJSIP/6007-00000000", "press_2") in new stack
-- <PJSIP/6007-00000000> Playing 'press_2.gsm' (language 'en')
-- Executing [100@from-internal:5] WaitExten("PJSIP/6007-00000000", "") in new stack
-- Executing [10@from-internal:1] Goto("PJSIP/6007-00000000", "select-option,1,1") in new stack
-- Goto (select-option,1,1)
```

Figure 39 Results from the Asterisk console

- Since the input is 2, the flow of execution will lead to the following part of the dial-plan. In this dial-plan, the callerID will be captured and stored in a variable, so will the current time. These will be used to name the individual audio files that will be recorded.

```
-- Executing [100@from-internal:1] Answer("PJSIP/6007-00000000", "") in new stack
> 0x7f01e40200f0 -- Strict RTP learning after remote address set to: 127.0.0.1:51548
> 0x7f01e40200f0 -- Strict RTP switching to RTP target address 127.0.0.1:51548 as source
-- Executing [100@from-internal:2] Playback("PJSIP/6007-00000000", "welcome") in new stack
-- <PJSIP/6007-00000000> Playing 'welcome.gsm' (language 'en')
> 0x7f01e40200f0 -- Strict RTP learning complete - Locking on source address 127.0.0.1:51548
-- Executing [100@from-internal:3] Playback("PJSIP/6007-00000000", "press_1") in new stack
-- <PJSIP/6007-00000000> Playing 'press_1.gsm' (language 'en')
-- Executing [100@from-internal:4] Playback("PJSIP/6007-00000000", "press_2") in new stack
-- <PJSIP/6007-00000000> Playing 'press_2.gsm' (language 'en')
-- Executing [100@from-internal:5] WaitExten("PJSIP/6007-00000000", "") in new stack
-- Executing [2@from-internal:1] Goto("PJSIP/6007-00000000", "select-option,2,1") in new stack
-- Goto (select-option,2,1)
-- Executing [2@select-option:1] Answer("PJSIP/6007-00000000", "") in new stack
-- Executing [2@select-option:2] Playback("PJSIP/6007-00000000", "select_2") in new stack
-- <PJSIP/6007-00000000> Playing 'select_2.gsm' (language 'en')
-- Executing [2@select-option:3] Set("PJSIP/6007-00000000", "CURRENT_TIME=20230823-223402") in new stack
-- Executing [2@select-option:4] Set("PJSIP/6007-00000000", "CALLER_ID=6007") in new stack
-- Executing [2@select-option:5] Set("PJSIP/6007-00000000", "i=1") in new stack
-- Executing [2@select-option:6] Playback("PJSIP/6007-00000000", "prompted_question") in new stack
-- <PJSIP/6007-00000000> Playing 'prompted_question.gsm' (language 'en')
-- Executing [2@select-option:7] System("PJSIP/6007-00000000", "mkdir -p /home/setenane/Desktop/SV_for_Password_Reset/verification/6007")
in new stack
-- Executing [2@select-option:8] Playback("PJSIP/6007-00000000", "question_1") in new stack
-- <PJSIP/6007-00000000> Playing 'question_1.gsm' (language 'en')
```

```
exten => 2,1,Answer()
same => n,Playback(select_2)
same => n,Set(CURRENT_TIME=${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
same => n,Set(CALLER_ID=${CALLERID(num)})
same => n,Set(i=1)

same => n,Playback(prompted_question)

; Create a new directory with the caller ID as the directory name
same => n,System(mkdir -p /home/setenane/Desktop/SV_for_Password_Reset/verification/${CALLER_ID})

; Loop to ask questions and record responses
same => n(loop),Playback(question_${i}) ; Playback the question message for the current iteration

; Set the recording duration based on the question number (i)
same => n,Set(RECORDING_DURATION=${IF($[${i} == 1]?20:${IF($[${i} == 2]?20:${IF($[${i} == 3]?30:${IF($[${i} == 4]?35:40)}}))}})

same => n,Set(RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/verification/${CALLER_ID}/${CALLER_ID}-${CURRENT_TIME}-${i}.wav)
same => n,Playback(beep) ; Play a beep sound before recording starts
same => n,Record(${RECORDED_FILE},${RECORDING_DURATION},10,16000)
same => n,Set(i=${i}+1)
same => n,GotoIf($[${i} < 6]?loop:end)

same => n(end),Playback(please_wait_processing)

; Execute the enrollment.py script using a system call, passing the callerID as an argument
same => n,System(/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/verification.py ${CALLER_ID})

same => n,System(/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/Verify_User.py ${CALLER_ID})

same => n,Set(VERIFICATION_RESULT=${SHELL(/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/Verify_User.py ${CALLER_ID})})
same => n,Verbose(VERIFICATION_RESULT is ${VERIFICATION_RESULT})

same => n,ExecIf("${VERIFICATION_RESULT}" > "0.5"?Goto(password_reset))
```

Figure 40 Capturing the CallerID and the current time

6. During the execution of the dial-plan, a new directory will be created, this directory will be used to store the caller's audio files.

```
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/verification/6007$ ls
6007-20230823-224101-1.wav 6007-20230823-224101-2.wav 6007-20230823-224101-3.wav 6007-20230823-224101-4.wav 6007-20230823-224101-5.wav
setenane@setenane-HP-250-G7-Notebook-PC:~/Desktop/SV_for_Password_Reset/verification/6007$
```

*Figure 41 A new directory created*

7. A loop is used to iterate over 5 questions that have been pre-recorded, the caller will be prompted to provide answers to these questions and the responses will be recorded.

```
-- Executing [2@select-option:1] Answer("PJSIP/6007-00000000", "") in new stack
-- Executing [2@select-option:2] Playback("PJSIP/6007-00000000", "select_2") in new stack
-- <PJSIP/6007-00000000> Playing 'select_2.gsm' (language 'en')
-- Executing [2@select-option:3] Set("PJSIP/6007-00000000", "CURRENT_TIME=20230823-223402") in new stack
-- Executing [2@select-option:4] Set("PJSIP/6007-00000000", "CALLER_ID=6007") in new stack
-- Executing [2@select-option:5] Set("PJSIP/6007-00000000", "i=1") in new stack
-- Executing [2@select-option:6] Playback("PJSIP/6007-00000000", "prompted_question") in new stack
```

*Figure 42 Responses from the question recorded*

8. After recording, the audio files will be stored in the created directory. Using the callerID, current time and an index,i.

```
/6007-20230823-224101-4.wav") in new stack
-- Executing [2@select-option:11] Playback("PJSIP/6007-00000001", "beep") in new stack
-- <PJSIP/6007-00000001> Playing 'beep.gsm' (language 'en')
-- Executing [2@select-option:12] Record("PJSIP/6007-00000001", "/home/setenane/Desktop/SV_for_Password_Reset/verification/6007/6007-20230823-224101-4.wav,35,10,16000") in new stack
```

*Figure 43 Storing audio files*

9. The caller will then be informed that there is some processing happening, while the caller waits, the dial-plan will execute two scripts. Firstly, verification.py will be executed, the dial-plan passes the callerID to the script, this script will load the recorded audio from the created directory, which uses callerID as its name. Then from the recorded audio files, relevant features are extracted, MFCCs.

```
-- Executing [2@select-option:15] Playback("PJSIP/6007-00000001", "please_wait_processing") in new stack
-- <PJSIP/6007-00000001> Playing 'please_wait_processing.gsm' (language 'en')
-- Executing [2@select-option:16] System("PJSIP/6007-00000001", "/usr/bin/python3 /home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/verification.py 6007") in new stack
Feature extraction and saving completed.
```

*Figure 44 Relevant features extracted from the audio files*

10. The second script to be executed is, `Verify_User.py`, again the `callerID` is given to the script as an argument, the script will load the extracted features using the `callerID`, to locate features specific to the caller. Using the loaded features, caller's specific model and universal model. Then the caller is informed that they have been verified successfully. The caller is then allowed to enter a new password into the dial pad on the SIP phone. After entering the new password, it is saved to a text file along with the `callerID`.

```
7 on new stack
VERIFICATION_RESULT is 6.795338990477418
Speaker is Verified

-- Executing [2@select-option:20] ExecIf("PJSIP/6007-00000000", "1?Goto(password_reset)") in new stack
-- Goto (select-option,2,22)
-- Executing [2@select-option:22] Playback("PJSIP/6007-00000000", "reset_password_prompt") in new stack
-- <PJSIP/6007-00000000> Playing 'reset_password_prompt.gsm' (language 'en')
-- Executing [2@select-option:23] Read("PJSIP/6007-00000000", "code_input,enter_code,4") in new stack
-- Accepting a maximum of 4 digits.
-- <PJSIP/6007-00000000> Playing 'enter_code.gsm' (language 'en')
-- User entered '5678'
-- Executing [2@select-option:24] Verbose("PJSIP/6007-00000000", "1, The entered code is: 5678") in new stack
The entered code is: 5678
-- Executing [2@select-option:25] Set("PJSIP/6007-00000000", "new_password=5678") in new stack
-- Executing [2@select-option:26] System("PJSIP/6007-00000000", "echo "6007 5678" >> /home/setenane/Desktop/SV_for_Password_Reset/password
s.txt") in new stack
-- Executing [2@select-option:27] Playback("PJSIP/6007-00000000", "verification_failed_prompt") in new stack
```

*Figure 45 Informing the user that is verified successfully*



*Figure 46 Shows that the user has successfully changed their password*

#### 4.3.2.1 Results and Discussions from the Experiment above

##### 4.3.2.1.1 Results

- a. When a caller placed a call, the call was answered,
- b. The caller then gives input to the dial-plan to indicate the caller wants to get verified using the service.
- c. The callerID was captured and used to create a directory that will store the caller's recorded audio files.
- d. The caller is asked questions and the caller's responses are stored in the specified directory.
- e. The script verification.py executed in the dial-plan, this script was able to load the audio files from their stored location, and extract relevant features (MFCCs) from the caller's audio files. The extracted features are then stored appropriately.
- f. The script Verify\_User.py executed in the dial-plan is able to load the features stored and use them along with caller's specific model and universal background model to verify the speaker then the caller is allowed to reset their password using the dial pad.
- g. The new password was successfully written to a text file along with callers' callerID.

##### 4.3.2.1.2 Discussions

Using the different technologies and tools, we were able to achieve verifying a caller using the system. By recording their audio files and storing them, extracting the necessary features from the audio recordings and using them with the caller's specific model and UBM to verify the caller to allow the caller to reset their password.



## Chapter 5. Conclusions and Future Work

### 5.1 Conclusions

#### 5.1.1 Concluding on results from Enrollment

From the call when is answered, the caller requests for enrollment whereby the caller's ID is captured and a directory is created for storing their audio files, the caller then responds to the questions and their responses are stored. We have a script "enrollment.py" that loads the audio files and extracts features (MFCCs) and then stores them. We also have another script "train\_UserGMM.py" that loads the extracted features and trains the Gaussian Mixture Model and then save the trained model.

#### 5.1.2 Concluding on results from Verification

When a call is placed, the caller provides input indicating their desire for verification. The caller's ID is captured and used to create a directory for storing recorded files. The caller is then asked questions and their responses are stored in this directory. We have a script called "verification.py" that is executed to extract relevant features (MFCCs) from the caller's audio files and then storing them appropriately. We also have another script "Verify\_User.py" that verifies the speaker based on the features extracted and the specific user model for each caller allowing them to reset their password. The system is able to successfully reset the callers' password by writing it to a text file.

Showing that we have achieved our objectives.

```
7:30 AM - 0000000000
VERIFICATION_RESULT is 6.795338990477418
Speaker is Verified

-- Executing [2@select-option:20] ExecIf("PJSIP/6007-00000000", "1?Goto(password_reset)") in new stack
-- Goto (select-option,2,22)
-- Executing [2@select-option:22] Playback("PJSIP/6007-00000000", "reset_password_prompt") in new stack
-- <PJSIP/6007-00000000> Playing 'reset_password_prompt.gsm' (language 'en')
-- Executing [2@select-option:23] Read("PJSIP/6007-00000000", "code_input,enter_code,4") in new stack
-- Accepting a maximum of 4 digits.
-- <PJSIP/6007-00000000> Playing 'enter_code.gsm' (language 'en')
-- User entered '5678'
-- Executing [2@select-option:24] Verbose("PJSIP/6007-00000000", "1, The entered code is: 5678") in new stack
The entered code is: 5678
-- Executing [2@select-option:25] Set("PJSIP/6007-00000000", "new_password=5678") in new stack
-- Executing [2@select-option:26] System("PJSIP/6007-00000000", "echo "6007 5678" >> /home/setenane/Desktop/SV_for_Password_Reset/password
s.txt") in new stack
-- Executing [2@select-option:27] Playback("PJSIP/6007-00000000", "verification_failed_prompt") in new stack
```

Figure 47 Showing that the caller is verified successfully

## 5.2 Future work

Issues that were essential in the project but out of scope of the project:

Issue	Definition
User Interface Design	As the project focuses on implementing Speaker Verification for password reset, the design and aesthetics of the user interface might be considered out of scope but yet considered as an important feature that ensures a user-friendly that enable in enhancing the overall user experience.
Accessibility	The system should be accessible to all users that include those with impairments. <sup>[9]</sup>
Scalability	A lot of users must be accommodated by the system. This requires making sure the system can correctly validate a large number of different voices in addition to the technical concerns of managing huge amounts of traffic. <sup>[8]</sup>
Legal and Ethical Considerations	The usage of biometric data, particularly voice data, is subject to legal and ethical issues. These factors might not fall under the project's purview, yet they are crucial to its long-term success. <sup>[10]</sup>
Privacy and Security	Even if the project's main objective is speaker verification, it's critical to make sure the system is safe and respects user privacy. This involves securing voice data from illegal access and making sure that voice impersonation or artificial voices can't readily mislead the system. <sup>[11]</sup>

*Table 11 Issues essential but out of scope in the project*

Future-relevant information: Machine learning and AI developments are continuously enhancing speech verification systems' accuracy. For instance, neural networks and deep learning approaches can enhance the system's capacity to recognize various voices. Future systems will also need to be able to discriminate between real and fake voices in order to prevent cheating as voice synthesis technology advances. <sup>[12]</sup>

As “Speaker Verification for password reset” involves several steps including data collection, feature extraction, model training and system integration, so given additional six months, the project could be developed as:

<b>Process</b>	<b>Estimate time (months)</b>	<b>Definition</b>
Data collection	1-2	Collect a large amount of voice data from different group of individuals that include what users might say during the verification process.
Feature extraction	1-2	Extract the features from the voice data that will be used to differentiate between different speakers.
Model training	2-3	Use of the machine learning algorithms (support vector machines, deep learning, gaussian mixture models) to train a model that can accurately identify each individual based on their voice.
System integration	1-2	Integrate the speaker verification model into the password reset system. This would involve developing an interface where users can speak into a microphone, as well as backend systems that can process the voice data and verify the user's identity.

*Table 12 Speaker Verification processes with their estimate time*

Additional features and functionalities to be recommended to make the system more complete, useful, usable and more secure:

<b>Feature or Functionality</b>	<b>Definition</b>
Multi-factor Authentication (MFA)	Increase the security of the system by combining the speaker verification with other forms of authentication such as fingerprint scan.
Noise Reduction	Include noise reduction algorithms in the system to improve the system's performance in noisy environments.
Adaptive Learning	Design the system that will learn and adapt to a user's voice over time thus improving its accuracy.
Voice Activity Detection	Implement a system that will detect when the user is speaking and automatically start recording thus making the system more user-friendly.
Anti-spoofing Measures	Implement measures to prevent voice spoofing attempts, such as liveness detection (which can tell if the speech is coming from a live person or a recording) or detect anomalies (which can find odd patterns in the voice data that can indicate a spoofing effort).
User Interface Design	Make sure the user interface is simple to use, with instructions that are clear to understand.
Privacy Measures	Ensure that voice data is stored and processed securely, with strong encryption and strict access controls, to protect users' privacy.

*Table 13 Additional features and functionalities*

## Chapter 6. References

1. <https://www.trustradius.com/authentication-systems>
2. Nilu Singh, Alka Agrawal, Raees Ahmad Khan., 2015. A Critical Review on Automatic Speaker Recognition.. *Science Journal of Circuits, Systems*, Vol. 4(doi: 10.11648/j.cssp.20150402.12), pp. pp. 14-17. And [https://www.researchgate.net/publication/281618217\\_A\\_Critical\\_Review\\_on\\_Automatic\\_Speaker\\_Recognition](https://www.researchgate.net/publication/281618217_A_Critical_Review_on_Automatic_Speaker_Recognition)
3. <https://agilemanifesto.org/principles.html>
4. <https://scrumguides.org/scrum-guide.html>
5. Mehta, N., 2023. *Agile Methodology*. [Online] Available at: <https://www.linkedin.com/pulse/agile-methodology-nitish-mehta> [Accessed 19 May 2023].
6. [https://www.researchgate.net/figure/A-block-diagram-of-a-text-independent-speaker-verification-system\\_fig1\\_268277174](https://www.researchgate.net/figure/A-block-diagram-of-a-text-independent-speaker-verification-system_fig1_268277174) and <https://maelfabien.github.io/machinelearning/Speech1/#>
7. <https://www.google.com/search?client=opera&q=pbx&sourceid=opera&ie=UTF-8&oe=UTF-8>
8. <https://stefanini.com/en/insights/articles/voice-biometrics-the-sound-of-security-in-the-age-of-ai>
9. <https://link.springer.com/article/10.1007/s41870-023-01224-8>
10. <https://www.microsoft.com/en-us/research/group/speech-research-team/>
11. [https://www.researchgate.net/publication/6300216\\_Ethical\\_and\\_social\\_implications\\_of\\_biometric\\_identification\\_technology](https://www.researchgate.net/publication/6300216_Ethical_and_social_implications_of_biometric_identification_technology)
12. <https://link.springer.com/article/10.1007/s10772-019-09665-y>

# Chapter 7. Appendices

## 7.1 Appendix A

### 7.1.1 List of Abbreviations

1. MFA: Multi-Factor Authentication
2. PINs: Personal Identification Numbers
3. USSD: Unstructured Supplementary Service Data
4. ER: Entity Relationship
5. SMS: Short Message/Messaging Service
6. SIM: Subscriber Identity Module
7. PBX: Private Branch Exchange
8. SIP phone: Session Initial Protocol phone
9. AI: Artificial Intelligence
- 10.FR: False Rejection
- 11.FA: False Acceptance
- 12.MFCCS: Mel-Frequency Cepstral Coefficients
- 13.UBM: Universal Background Model
- 14.GMM: Gaussian Mixture Model
- 15.RAM: Random Access Memory

## 7.2 Appendix B

### 7.2.1 System Manual

The Speaker Verification for Password Reset system is designed to allow users to reset their password through voice recognition technology. The system works by capturing a voice sample during the user enrollment process and storing it in a database. When a user requests a password reset, the system will prompt the user to provide a voice sample to compare against the registered sample.

The system uses machine learning algorithms to analyze the voice samples and determine if they match. If the samples match, the user will be able to reset their password. If the samples do not match, the user will not be able to reset their password.

Note: To ensure the accuracy of the system, it is recommended to use a high-quality microphone and to speak clearly during the registration and password reset processes.

#### 7.2.1.1 Requirements

1. Asterisk PBX:
2. **Operating System:**
  - a. Linux (recommended distributions include CentOS, Ubuntu, Debian)
  - b. Processor: 1 GHz or faster
  - c. RAM: 1 GB or more
  - d. Disk Space: At least 20 GB of free space
  - e. Network: Ethernet interface
3. **Zoiper 5:**
  - a. Operating System: Windows 7 or later, macOS 10.10 or later, Linux (various distributions).
  - b. Processor: 1 GHz or faster.

- c. RAM: 512 MB or more.
- d. Network: Broadband internet connection for VoIP calls.
- e. Microphone and Speakers/Headset: Required for making/receiving calls.
- f. Webcam (for video calls).

#### **4. Octave:**

- a. Operating System: Windows, macOS, Linux.
- b. Processor: 1 GHz or faster.
- c. RAM: 2 GB or more.
- d. Disk Space: Several GBs for installation and data storage.
- e. For advanced use, higher computational resources might be needed.
- f. scikit-learn.

#### **5. Python:**

- a. Versions 3.6 and above.
- b. Operating System: Windows, macOS, Linux.
- c. Dependencies: NumPy, SciPy.
- d. Processor: 1 GHz or faster.
- e. RAM: 1 GB or more (more for resource-intensive tasks).
- f. Disk Space: Depending on the applications and libraries you install; a few GBs are usually sufficient.

#### 7.2.1.2 Installation

##### **1. Asterisk PBX:**

Install Linux: Choose a supported Linux distribution (e.g., CentOS, Ubuntu) and follow their installation instructions.

Install Dependencies: Open a terminal and install necessary dependencies using package manager commands specific to your distribution.

Download Asterisk: Download the Asterisk source code from the official website.



**Compile and Install:** Extract the downloaded source code, navigate to the extracted directory, and run the compilation and installation commands.

**Configure Asterisk:** Edit configuration files as needed for your setup (e.g., sip.conf, extensions.conf).

**Start Asterisk:** Start the Asterisk service using the appropriate command.

**Test:** Use a softphone or SIP client to test your Asterisk installation.

## **2. Zoiper 5:**

**Download Installer:** Visit the Zoiper website and download the installer for your operating system.

**Run Installer:** Run the downloaded installer and follow the on-screen instructions.

**Launch Zoiper:** Once installed, launch Zoiper.

**Set Up Account:** Configure your SIP or VoIP account within Zoiper by providing the necessary credentials.

**Test:** Make a test call to ensure Zoiper is working correctly.

## **3. Octave:**

**Install Octave:** Use your package manager (on Linux) or download the installer (on Windows/macOS) from the Octave website.

**Run Octave:** Launch Octave from the terminal or the provided shortcut.

**Start Using:** Octave's command-line interface will open. You can start running Octave commands and scripts.

## **4. scikit-learn:**

**Python Environment:** Ensure you have a working Python environment (Python 3.6 or above) and pip installed.

**Install scikit-learn:** Open a terminal and run `pip install scikit-learn` to install the library and its dependencies.

## 5. Python:

**Download Python:** Download the Python installer for your operating system from the official Python website.

**Run Installer:** Run the downloaded installer and follow the installation prompts.

**Verify Installation:** Open a terminal and run `python --version` to verify the installation.

**Install Packages:** Use `pip`, the Python package manager, to install additional packages like `scikit-learn`. For example, `pip install scikit-learn`.

### 7.2.1.3 Usage of the System

1. Asterisk 16.4.0
2. GNU Octave, version 6.4.0
3. Python3

For a Systems operator to use the Speaker Verification for Password Reset System, the operator would need to install the above-mentioned software correctly, in accordance with their available systems and infrastructures. The operator can use the scripts provided in the Appendix to set up the system. When implemented, the code will set up the functions and processes that make up the speaker verification system.

The operator can then use the system to verify speakers. However, there are modifications to be made to the scripts such as the locations of directories that will store the audio files, features, models, etc.

### 7.2.1.4 Details on Hardware and Software Configuration and Troubleshooting

Hardware to configure would be the PC on which Asterisk will be installed, as Asterisk is fully functional only on Linux systems such as Ubuntu.

Again, the Asterisk software would need to be configured, the provided scripts in the Appendix can be used to configure Asterisk software, namely `extensions.conf`, `sip.conf` and `pjsip.conf`. They can be adjusted to suit the desires of another systems operator.

## 7.3 Appendix C

### 7.3.1 User Manual

The Speaker Verification for Password Reset system allows you to reset your password through voice recognition technology. To use the system, you need to follow these steps:

1. First do a call and you will be asked to press either 1 for enrollment or 2 for password change/reset.
2. For enrollment process, you will be prompted to provide a voice sample that will be captured to create a model then stored in the database. To ensure the accuracy of the system, speak clearly and avoid background noise.
3. For password change/reset, you will also be guided with instructions and be prompt to provide voice sample that will be compared with the stored samples from the created model stored in the database. If the system is unable to match your voice sample with the registered/stored sample, you will need to start again from the beginning of the process or you not allowed to reset your password. If the voice sample is successfully matched, you will be prompted to enter a new password. Which will be captured then saved.

Note: Speak clearly and follow the instructions given on each of the processes.

## 7.4 Appendix D

### 7.4.1 Code Snippet

#### 7.4.1.1 Sip.conf

Below is a description of chan.sip, it is used to create contacts that make calls to the Asterisk PBX.

```
[general]
```

```
context=default
```

```
[6001]
```

```
type=friend
```

```
context=from-internal
```

```
host=dynamic
```

```
secret=1234
```

```
;disallow=all
```

```
allow=all
```

```
[6002]
```

```
type=friend
```

```
context=from-internal
```

```
host=dynamic
```

```
secret=1234
```

```
;disallow=all
```

```
allow=all
```

```
[6003]
```

```
type=friend
```

```
context=from-internal
```

```
host=dynamic
```

```
secret=1234
```

```
;disallow=all
```

allow=all

[6004]

type=friend

context=from-internal

host=dynamic

secret=1234

;disallow=all

allow=all

[6005]

type=friend

context=from-internal

host=dynamic

secret=1234

;disallow=all

allow=all

[6006]

type=friend

context=from-internal

host=dynamic

secret=1234

;disallow=all

allow=all

[6007]

type=friend

context=from-internal

host=dynamic

secret=1234

;disallow=all

allow=all

#### 7.4.1.2 Pjsip.conf

Below is the configuration of chan.pjsip, it is used to configure contacts that can make calls to the Asterisk PBX which then enables the caller to enroll for the Speaker Verification for Password Reset system and verify the caller to reset their password.

```
[transport-udp]
```

```
type=transport
```

```
protocol=udp
```

```
bind=0.0.0.0
```

```
[6001]
```

```
type=endpoint
```

```
context=from-internal
```

```
;disallow=all
```

```
allow=all
```

```
auth=6001
```

```
aors=6001
```

```
;expiration=3600 ; Set expiration time to 3600 seconds (1 hour)
```

```
[6001]
```

```
type=auth
```

```
auth_type=userpass
```

```
password=1234
```

```
username=6001
```

```
[6001]
```

```
type=aor
```

```
max_contacts=1
```

```
[6002]
```

```
type=endpoint
```

```
context=from-internal
```

```
;disallow=all
```

```

allow=all
auth=6002
aors=6002
;expiration=3600 ; Set expiration time to 3600 seconds (1 hour)
[6002]
type=auth
auth_type=userpass
password=1234
username=6002

[6002]
type=aor
max_contacts=1

```

#### 7.4.1.3 Extensions.conf

The following code is a Dial-plan, the dial-plan is simply instructions telling Asterisk what to do with a call. When a call is initiated, using the appropriate extension, the following lines will be executed during the call.

```

[from-internal]
exten => t,1,NoOp()
exten => e,1,NoOp()

exten => 100,1,Answer()
same => n,Playback(welcome)
same => n,Playback(press_1)
same => n,Playback(press_2)
same => n,WaitExten()

exten => 1,1,Goto(select-option,1,1)
exten => 2,1,Goto(select-option,2,1)

```



#### 7.4.1.4 UBM\_GMM.m

The code below is used to generate Mel-Frequency Cepstral Coefficients, by extracting MFCCs from the audio files located in the specified directory, the extracted features are stored to be used for training a Universal Background Model.

```
% Define variables
```

```
Tw = 25;          % analysis frame duration (ms)
Ts = 10;          % analysis frame shift (ms)
alpha = 0.97;     % preemphasis coefficient
M = 20;           % number of filterbank channels
C = 12;           % number of cepstral coefficients
L = 22;           % cepstral sine lifter parameter
LF = 300;         % lower frequency limit (Hz)
HF = 3700;        % upper frequency limit (Hz)
```

```
% Set the directory containing the audio files
```

```
rootDir = '/home/setenane/Desktop/SV_for_Password_Reset/UBM_training';
```

```
% Initialize a matrix to store MFCC features
```

```
allMFCCs = [];
```

```
% Recursively search for all .wav files in sub-folders
```

```
audioFiles = dir(fullfile(rootDir, '**', '*.wav'));
```

```
% Loop through each audio file
```

```
for i = 1:length(audioFiles)
```

```
    % Construct the full path to the audio file
```

```
    filePath = fullfile(audioFiles(i).folder, audioFiles(i).name);
```

```
    % Read speech samples and sampling rate from the file
```

```
    [speech, fs] = audioread(filePath);
```

```
    % Feature extraction
```

```

[MFCCs, ~, ~] = mfcc(speech, fs, Tw, Ts, alpha, @hamming, [LF HF], M, C+1, L);

% Append the extracted MFCCs to the matrix
allMFCCs = [allMFCCs, MFCCs]; % Use comma to concatenate columns
end

% Save the extracted features to a plain text file with .dat extension
savePath = '/home/setenane/Desktop/SV_for_Password_Reset/UBM_training/UBM_GMM.dat';
dlmwrite(savePath, allMFCCs, ' ');

disp('Feature extraction and saving completed.');
```

% EOF

#### 7.4.1.5 train\_UBM.py

Script below loads the extracted features from the location '/home/setenane/Desktop/SV\_for\_Password\_Reset/UBM\_training/UBM\_GMM.dat', then trains a Universal Background Model/Gaussian Mixture Model.

```

import numpy as np
from sklearn.mixture import GaussianMixture
import joblib

# Load the saved features
savePath = '/home/setenane/Desktop/SV_for_Password_Reset/UBM_training/UBM_GMM.dat'
allMFCCs = np.loadtxt(savePath)

# Train a Gaussian Mixture Model
gmm = GaussianMixture(n_components=64, covariance_type='diag', random_state=0)
gmm.fit(allMFCCs.T)

# Save the trained model
```

```

modelPath =
'/home/setenane/Desktop/SV_for_Password_Reset/Universal_Background_Model/GMM_UBM.
pkl'
joblib.dump(gmm, modelPath)

```

#### 7.4.1.6 extensions.conf (enrollment)

Below is a dial-plan used to enroll callers for the Speaker Verification for Password Reset System. Audio files will be recorded from the caller, stored in a location, the same audio files are processed to extract relevant features from them, the same features are used to train a caller specific model.

```

[select-option]
exten => 1,1,Answer()
same => n,Playback(select_1)
same => n,Set(CURRENT_TIME=${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
same => n,Set(CALLER_ID=${CALLERID(num)})
same => n,Set(i=1)

same => n,Playback(prompted_question)

; Create a new directory with the caller ID as the directory name
same => n,System(mkdir -p
/home/setenane/Desktop/SV_for_Password_Reset/enrollment/${CALLER_ID})

; Loop to ask questions and record responses
same => n(loop),Playback(question_${i}) ; Playback the question message for the current
iteration

; Set the recording duration based on the question number (i)
same => n,Set(RECORDING_DURATION=${IF($[${i}] == 1)?20:${IF($[${i}] == 2)?20:${IF($[${i}]
== 3)?30:${IF($[${i}] == 4)?35:40}})}}))

```

```

same =>
n,Set(RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/enrollment/${CALLER_ID}/${CALLER_ID}-${CURRENT_TIME}-${i}.wav)
same => n,Playback(beep) ; Play a beep sound before recording starts
same => n,Record(${RECORDED_FILE},${RECORDING_DURATION},10,16000)
same => n,Set(i=${i}+1)
same => n,GotoIf(${i} < 6)?loop:end

same => n(end),Playback(please_wait_processing)

; Execute the enrollment.py script using a system call, passing the callerID as an argument
same => n,System(/usr/bin/python3
/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/enrollment.py
${CALLER_ID})

same => n,System(/usr/bin/python3
/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/train_UserGMM.py
${CALLER_ID})

; Handle enrollment success
same => n,Playback(enrolled_successfully)

; Playback recorded successfully and thank you messages
same => n,Playback(thank_you)

; Hang up the call
same => n,Hangup()

```

#### 7.4.1.7 enrollment.py

Script below will be used to open Octave; Octave will execute the enrollment function using the caller's ID. The working directory will be added to the path, so that the function enrollment can be found and executed.

```

#!/usr/bin/env python3
import os
import subprocess
import sys # Import sys module to access command line arguments

# Set the login name and current working directory
working_directory = "/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc"

# Change the current working directory
os.chdir(working_directory)

# Get the caller's ID argument passed from the dialplan
if len(sys.argv) < 2:
    print("Error: Missing caller's ID argument.")
    sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Execute the enrollment.m script in Octave using the caller's ID
octave_command = f"octave --eval \"addpath('{working_directory}'); enrollment('{callerID}'); exit;\""

try:
    # Capture the output and error messages
    result = subprocess.run(octave_command, shell=True, capture_output=True, text=True)

except subprocess.CalledProcessError as e:
    print(f"Error executing Octave command: {e}")

```

#### 7.4.1.8 enrollment.m

Function below is used to extract MFCCs from the callers' audio files, using the caller's ID to locate a repository containing the callers' recorded audio files then extracting relevant features from them.

```
function enrollment(callerID)

% Define variables
Tw = 25;           % analysis frame duration (ms)
Ts = 10;           % analysis frame shift (ms)
alpha = 0.97;      % preemphasis coefficient
M = 20;            % number of filterbank channels
C = 12;            % number of cepstral coefficients
L = 22;            % cepstral sine lifter parameter
LF = 300;          % lower frequency limit (Hz)
HF = 3700;         % upper frequency limit (Hz)

% Set the directory containing the audio files
audioDir = ['/home/setenane/Desktop/SV_for_Password_Reset/enrollment/', callerID];

% List all audio files in the directory
audioFiles = dir(fullfile(audioDir, '*.wav'));

% Initialize a matrix to store MFCC features
allMFCCs = [];

% Loop through each audio file
for i = 1:length(audioFiles)
    % Construct the full path to the audio file
    filePath = fullfile(audioDir, audioFiles(i).name);

    % Read speech samples and sampling rate from the file
    [speech, fs] = audioread(filePath);
```

```

% Feature extraction
[MFCCs, ~, ~] = mfcc(speech, fs, Tw, Ts, alpha, @hamming, [LF HF], M, C+1, L);

% Append the extracted MFCCs to the matrix
allMFCCs = [allMFCCs, MFCCs];
end

% Save the extracted features to a plain text file with .dat extension
savePath = ['/home/setenane/Desktop/SV_for_Password_Reset/Enrollment_features/',
callerID, '.dat'];
dlmwrite(savePath, allMFCCs, ' ');

disp('Feature extraction and saving completed.');
```

end

#### 7.4.1.9 train\_UserGMM.Py

Script below will load the callers' extracted features, using the callerID to load the callers' specific file that contains the caller's audio features. The extracted features are then used to train a Gaussian Mixture Model, specific to the caller.

```

#!/usr/bin/env python3
import os
import subprocess
import sys # Import sys module to access command line arguments

# Set the login name and current working directory
working_directory = "/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc"

# Change the current working directory
os.chdir(working_directory)
```

```

# Get the callerID argument passed from the dialplan
if len(sys.argv) < 2:
    print("Error: Missing callerID argument.")
    sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Execute the enrollment.m script in Octave using the callerID
octave_command = f"octave --eval \"addpath('{working_directory}'); enrollment('{callerID}');
exit;\""

try:
    # Capture the output and error messages
    result = subprocess.run(octave_command, shell=True, capture_output=True, text=True)

except subprocess.CalledProcessError as e:
    print(f"Error executing Octave command: {e}")
import numpy as np
from sklearn.mixture import GaussianMixture
import joblib
import sys

# Get the callerID argument passed from the command line
if len(sys.argv) < 2:
    print("Error: Missing callerID argument.")
    sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Load the saved features
savePath =
f'/home/setenane/Desktop/SV_for_Password_Reset/Enrollment_features/{callerID}.dat'
allMFCCs = np.loadtxt(savePath)

```



```

# Train a Gaussian Mixture Model
gmm = GaussianMixture(n_components=64, covariance_type='diag', random_state=0)
gmm.fit(allMFCCs.T)

# Save the trained model
modelPath = f'/home/setenane/Desktop/SV_for_Password_Reset/User_models/{callerID}.pkl'
joblib.dump(gmm, modelPath)

# Store the enrollment result in a variable
ENROLLMENT_RESULT = 'Enrolled Successfully'

# Remove leading and trailing whitespace
ENROLLMENT_RESULT = ENROLLMENT_RESULT.strip()

# Print the enrollment result
print(ENROLLMENT_RESULT)

# Return the enrollment result to the dialplan
sys.stdout.flush()

```

#### 7.4.1.10 mfcc.m

Function below is used by the scripts, UBM\_GMM.m, enrollment.m, and verification.m. The function is used to extract relevant features from audio files, these features, MFCCs, are then used to train models.

```

function [ CC, FBE, frames ] = mfcc( speech, fs, Tw, Ts, alpha, window, R, M, N, L )
% MFCC Mel frequency cepstral coefficient feature extraction.
%
% MFCC(S,FS,TW,TS,ALPHA,WINDOW,R,M,N,L) returns mel frequency
% cepstral coefficients (MFCCs) computed from speech signal given
% in vector S and sampled at FS (Hz). The speech signal is first
% preemphasised using a first order FIR filter with preemphasis

```

```

% coefficient ALPHA. The preemphasised speech signal is subjected
% to the short-time Fourier transform analysis with frame durations
% of TW (ms), frame shifts of TS (ms) and analysis window function
% given as a function handle in WINDOW. This is followed by magnitude
% spectrum computation followed by filterbank design with M triangular
% filters uniformly spaced on the mel scale between lower and upper
% frequency limits given in R (Hz). The filterbank is applied to
% the magnitude spectrum values to produce filterbank energies (FBEs)
% (M per frame). Log-compressed FBEs are then decorrelated using the
% discrete cosine transform to produce cepstral coefficients. Final
% step applies sinusoidal lifter to produce liftered MFCCs that
% closely match those produced by HTK [1].
%
% [CC,FBE,FRAMES]=MFCC(...) also returns FBEs and windowed frames,
% with feature vectors and frames as columns.
%
% This framework is based on Dan Ellis' rastamat routines [2]. The
% emphasis is placed on closely matching MFCCs produced by HTK [1]
% (refer to p.337 of [1] for HTK's defaults) with simplicity and
% compactness as main considerations, but at a cost of reduced
% flexibility. This routine is meant to be easy to extend, and as
% a starting point for work with cepstral coefficients in MATLAB.
% The triangular filterbank equations are given in [3].
%
% Inputs
%
%     S is the input speech signal (as vector)
%
%
%     FS is the sampling frequency (Hz)
%
%
%     TW is the analysis frame duration (ms)
%
%
%     TS is the analysis frame shift (ms)
%
%
%     ALPHA is the preemphasis coefficient

```

```

%
% WINDOW is a analysis window function handle
%
% R is the frequency range (Hz) for filterbank analysis
%
% M is the number of filterbank channels
%
% N is the number of cepstral coefficients
%   (including the 0th coefficient)
%
% L is the liftering parameter
%
% Outputs
% CC is a matrix of mel frequency cepstral coefficients
%   (MFCCs) with feature vectors as columns
%
% FBE is a matrix of filterbank energies
%   with feature vectors as columns
%
% FRAMES is a matrix of windowed frames
%   (one frame per column)
%
% Example
% Tw = 25;      % analysis frame duration (ms)
% Ts = 10;      % analysis frame shift (ms)
% alpha = 0.97; % preemphasis coefficient
% R = [ 300 3700 ]; % frequency range to consider
% M = 20;      % number of filterbank channels
% C = 13;      % number of cepstral coefficients
% L = 22;      % cepstral sine lifter parameter
%
% % hamming window (see Eq. (5.2) on p.73 of [1])
% hamming = @(N)(0.54-0.46*cos(2*pi*[0:N-1]./(N-1)));
%

```

```

%      % Read speech samples, sampling rate and precision from file
%      [ speech, fs, nbits ] = wavread( 'sp10.wav' );
%
%      % Feature extraction (feature vectors as columns)
%      [ MFCCs, FBEs, frames ] = ...
%          mfcc( speech, fs, Tw, Ts, alpha, hamming, R, M, C, L );
%
%      % Plot cepstrum over time
%      figure('Position', [30 100 800 200], 'PaperPositionMode', 'auto', ...
%          'color', 'w', 'PaperOrientation', 'landscape', 'Visible', 'on' );
%
%      imagesc( [1:size(MFCCs,2)], [0:C-1], MFCCs );
%      axis( 'xy' );
%      xlabel( 'Frame index' );
%      ylabel( 'Cepstrum index' );
%      title( 'Mel frequency cepstrum' );
%
%  References
%
%      [1] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D.,
%          Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D.,
%          Valtchev, V., Woodland, P., 2006. The HTK Book (for HTK
%          Version 3.4.1). Engineering Department, Cambridge University.
%          (see also: http://htk.eng.cam.ac.uk)
%
%      [2] Ellis, D., 2005. Reproducing the feature outputs of
%          common programs using Matlab and melfcc.m. url:
%          http://labrosa.ee.columbia.edu/matlab/rastamat/mfccs.html
%
%      [3] Huang, X., Acero, A., Hon, H., 2001. Spoken Language
%          Processing: A guide to theory, algorithm, and system
%          development. Prentice Hall, Upper Saddle River, NJ,
%          USA (pp. 314-315).
%

```

% See also EXAMPLE, COMPARE, FRAMES2VEC, TRIFBANK.

% Author: Kamil Wojcicki, September 2011

%% PRELIMINARIES

% Ensure correct number of inputs

if( nargin~= 10 ), help mfcc; return; end;

% Explode samples to the range of 16 bit shorts

if( max(abs(speech))<=1 ), speech = speech \* 2^15; end;

Nw = round( 1E-3\*Tw\*fs ); % frame duration (samples)

Ns = round( 1E-3\*Ts\*fs ); % frame shift (samples)

nfft = 2^nextpow2( Nw ); % length of FFT analysis

K = nfft/2+1; % length of the unique part of the FFT

%% HANDY INLINE FUNCTION HANDLES

% Forward and backward mel frequency warping (see Eq. (5.13) on p.76 of [1])

% Note that base 10 is used in [1], while base e is used here and in HTK code

hz2mel = @( hz )( 1127\*log(1+hz/700) ); % Hertz to mel warping function

mel2hz = @( mel )( 700\*exp(mel/1127)-700 ); % mel to Hertz warping function

% Type III DCT matrix routine (see Eq. (5.14) on p.77 of [1])

dctm = @( N, M )( sqrt(2.0/M) \* cos( repmat([0:N-1].',1,M) ...  
.\* repmat(pi\*([1:M]-0.5)/M,N,1) ) );

% Cepstral lifter routine (see Eq. (5.12) on p.75 of [1])

ceplifter = @( N, L )( 1+0.5\*L\*sin(pi\*[0:N-1]/L) );

## %% FEATURE EXTRACTION

% Preemphasis filtering (see Eq. (5.1) on p.73 of [1])

```
speech = filter( [1 -alpha], 1, speech ); % fvtool( [1 -alpha], 1 );
```

% Framing and windowing (frames as columns)

```
frames = vec2frames( speech, Nw, Ns, 'cols', window, false );
```

% Magnitude spectrum computation (as column vectors)

```
MAG = abs( fft(frames,nfft,1) );
```

% Triangular filterbank with uniformly spaced filters on mel scale

```
H = trifbank( M, K, R, fs, hz2mel, mel2hz ); % size of H is M x K
```

% Filterbank application to unique part of the magnitude spectrum

```
FBE = H * MAG(1:K,:); % FBE( FBE<1.0 ) = 1.0; % apply mel floor
```

% DCT matrix computation

```
DCT = dctm( N, M );
```

% Conversion of logFBEs to cepstral coefficients through DCT

```
CC = DCT * log( FBE );
```

% Cepstral lifter computation

```
lifter = ceplifter( N, L );
```

% Cepstral liftering gives liftered cepstral coefficients

```
CC = diag( lifter ) * CC; % ~ HTK's MFCCs
```

% EOF

#### 7.4.1.11 extensions.conf (verificaiton)

Below is a dial-plan used to verify callers using the Speaker Verification for Password Reset System. Audio files will be recorded from the caller, stored in a location, the same audio files are processed to extract relevant features from them using scripts executed in the following dial-plan. The dial-plan also perform verification using the user's specific model, universal background model and the extracted features.

```
exten => 2,1,Answer()
same => n,Playback(select_2)
same => n,Set(CURRENT_TIME=${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
same => n,Set(CALLER_ID=${CALLERID(num)})
same => n,Set(i=1)

same => n,Playback(prompted_question)

; Create a new directory with the caller ID as the directory name
same => n,System(mkdir -p
/home/setenane/Desktop/SV_for_Password_Reset/verification/${CALLER_ID})

; Loop to ask questions and record responses
same => n(loop),Playback(question_${i}) ; Playback the question message for the current
iteration

; Set the recording duration based on the question number (i)
same => n,Set(RECORDING_DURATION=${IF(${i} == 1)?20:${IF(${i} == 2)?20:${IF(${i}
== 3)?30:${IF(${i} == 4)?35:40}})})

same =>
n,Set(RECORDED_FILE=/home/setenane/Desktop/SV_for_Password_Reset/verification/${CAL
LER_ID}/${CALLER_ID}-${CURRENT_TIME}-${i}.wav)
same => n,Playback(beep) ; Play a beep sound before recording starts
same => n,Record(${RECORDED_FILE},${RECORDING_DURATION},10,16000)
same => n,Set(i=${i}+1)
```

```
same => n, GotoIf($[${i} < 6]?loop:end)
```

```
same => n(end), Playback(please_wait_processing)
```

```
; Execute the enrollment.py script using a system call, passing the callerID as an argument
```

```
same => n, System(/usr/bin/python3  
/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/verification.py  
${CALLER_ID})
```

```
same => n, System(/usr/bin/python3  
/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/Verify_User.py  
${CALLER_ID})
```

```
same => n, Set(VERIFICATION_RESULT=${SHELL(/usr/bin/python3  
/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc/Verify_User.py  
${CALLER_ID})})
```

```
same => n, Verbose(VERIFICATION_RESULT is ${VERIFICATION_RESULT})
```

```
same => n, ExecIf("${VERIFICATION_RESULT}" > "0.5"?Goto(password_reset))
```

```
same => n, Goto(verification_failed)
```

```
same => n(password_reset), Playback(reset_password_prompt)
```

```
same => n, Read(code_input, enter_code, 4)
```

```
same => n, Verbose(1, The entered code is: ${code_input})
```

```
; Store the 4-digit code in a variable called new_password
```

```
same => n, Set(new_password=${code_input})
```

```
; Write the caller_id and new_password to passwords.txt
```

```
same => n, System(echo "${CALLER_ID} ${new_password}" >>  
/home/setenane/Desktop/SV_for_Password_Reset/passwords.txt)
```

```
; Handle verification failure
```

```
same => n(verification_failed), Playback(verification_failed_prompt)
```



same => n, Hangup()

#### 7.4.1.12 verification.py

Script below will be used to open Octave; Octave will execute the verification function using the callerID. The working directory will be added to the path, so that the function verification can be found and executed.

```
#!/usr/bin/env python3
import os
import subprocess
import sys # Import sys module to access command line arguments

# Set the login name and current working directory
working_directory = "/home/setenane/Desktop/SV_for_Password_Reset/Code_Snippets/mfcc"

# Change the current working directory
os.chdir(working_directory)

# Get the callerID argument passed from the dialplan
if len(sys.argv) < 2:
    print("Error: Missing callerID argument.")
    sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Execute the enrollment.m script in Octave using the callerID
octave_command = f"octave --eval \"addpath('{working_directory}'); verification('{callerID}')";
exit;\""

try:
    # Capture the output and error messages
    result = subprocess.run(octave_command, shell=True, capture_output=True, text=True)
```

```
# Get the verification result from Octave's output
verification_result = result.stdout.strip()

# Print the verification result for Asterisk to capture
print(verification_result)
```

```
except subprocess.CalledProcessError as e:
    print(f"Error executing Octave command: {e}")
```

#### 7.4.1.13 verification.m

Function below is used to extract MFCCs from the callers' audio files, using the caller's ID to locate a repository containing the callers' recorded audio files then extracting relevant features from them.

```
function verification(callerID)
    % Define variables
    Tw = 25;          % analysis frame duration (ms)
    Ts = 10;          % analysis frame shift (ms)
    alpha = 0.97;     % preemphasis coefficient
    M = 20;            % number of filterbank channels
    C = 12;            % number of cepstral coefficients
    L = 22;            % cepstral sine lifter parameter
    LF = 300;          % lower frequency limit (Hz)
    HF = 3700;         % upper frequency limit (Hz)

    % Set the directory containing the audio files
    audioDir = ['/home/setenane/Desktop/SV_for_Password_Reset/verification/', callerID];

    % List all audio files in the directory
    audioFiles = dir(fullfile(audioDir, '*.wav'));

    % Initialize a matrix to store MFCC features
    allMFCCs = [];
```

```

% Loop through each audio file
for i = 1:length(audioFiles)
    % Construct the full path to the audio file
    filePath = fullfile(audioDir, audioFiles(i).name);

    % Read speech samples and sampling rate from the file
    [speech, fs] = audioread(filePath);

    % Feature extraction
    [MFCCs, ~, ~] = mfcc(speech, fs, Tw, Ts, alpha, @hamming, [LF HF], M, C+1, L);

    % Append the extracted MFCCs to the matrix
    allMFCCs = [allMFCCs, MFCCs];
end

% Save the extracted features to a plain text file with .dat extension
savePath = ['/home/setenane/Desktop/SV_for_Password_Reset/Verification_features/',
callerID, '.dat'];
dlmwrite(savePath, allMFCCs, ' ');

disp('Feature extraction and saving completed.');
```

end

#### 7.4.1.14 Verify\_User.py

```

import numpy as np
import joblib
import sys

# Get the callerID argument passed from the command line
if len(sys.argv) < 2:
```

```

print("Error: Missing callerID argument.")
sys.exit(1)

callerID = str(sys.argv[1]) # Convert the argument to a string

# Load the saved features
savePath =
f'/home/setenane/Desktop/SV_for_Password_Reset/Verification_features/{callerID}.dat'
allMFCCs = np.loadtxt(savePath)

# Load the universal background model
ubmModelPath =
'/home/setenane/Desktop/SV_for_Password_Reset/Universal_Background_Model/GMM_UBM.
.pkl'
ubmGmm = joblib.load(ubmModelPath)

# Load the user-specific model
userModelPath =
f'/home/setenane/Desktop/SV_for_Password_Reset/User_models/{callerID}.pkl'
userGmm = joblib.load(userModelPath)

# Perform speaker verification
ubmScores = ubmGmm.score_samples(allMFCCs.T)
userScores = userGmm.score_samples(allMFCCs.T)

# Set a threshold for verification
threshold = -1
verification = 0

difference = userScores.mean() - ubmScores.mean()

if difference > threshold:
    verification = 1
else:

```

```
verification = 0

if verification == 1:
    print(difference)
    print("Speaker is Verified")
else:
    print(difference)
    print("Verification failed")
```

## 7.5 Appendix E

### 7.5.1 Project Schedule

PROCESSES	START DATE	END DATE	DURATION(Days)
Idea proposal	12/5/2022	12/7/2022	2
Writing proposal document	12/9/2022	12/12/2022	3
Submitting the document	12/12/2022	12/13/2022	1
Proposal returned	12/13/2022	12/14/2022	1
Presentation of the Proposal	12/14/2022	12/15/2022	1
Developing front-end	12/24/2022	2/26/2023	64
Creating Database	3/12/2023	3/26/2023	14
Developing Speaker Verification	4/7/2023	5/3/2023	26
Testing Speaker Verification	5/9/2023	5/15/2023	6
Evaluation of the System	5/18/2023	5/27/2023	9
TOTAL NUMBER OF DAYS			127

*Figure 48 Project Schedule*

## 7.5.2 Gantt Chart

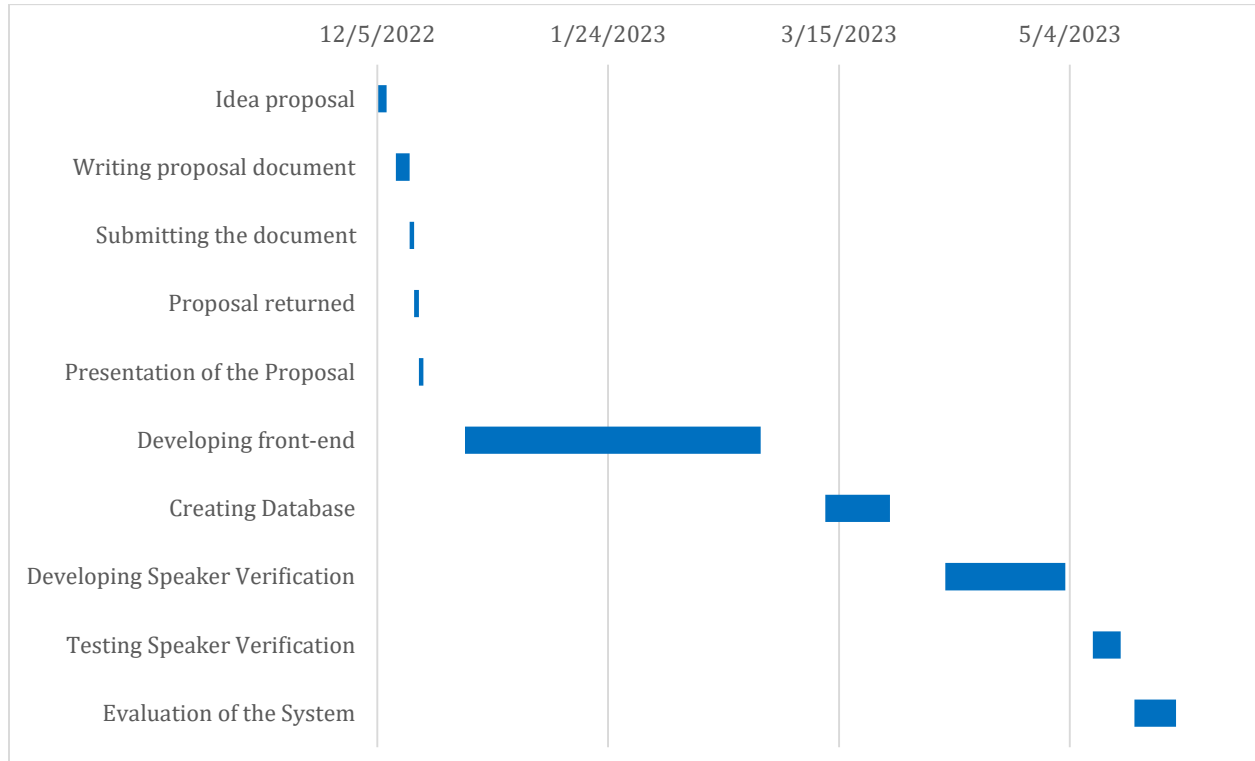


Figure 49 Gantt Chart