B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

From Sound to Security

By
Cormac Geraghty

April 27, 2025

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Abstract

This project presents an audio-based password generation and authentication system designed to address limitations in traditional password and biometric security methods. By extracting stable features from user-provided audio files and leveraging AI-driven password generation techniques, the system eliminates the need for password memorisation while maintaining strong security properties. Extracted features are processed through a cryptographic pipeline involving hashing, password generation via structured prompting of GPT-4o, and encrypted storage using PBKDF2 and Fernet encryption. Evaluation shows that generated passwords achieve an average entropy of 78.33 bits, with high resistance to brute-force and AI-based prediction attacks. Processing performance remains suitable for high-security domains, despite feature extraction representing the primary computational bottleneck. Comparative analysis demonstrates that the system provides stronger security and better privacy protection than traditional and biometric approaches, although reliance on external services and lack of formal user testing represent areas for future improvement. The project contributes a modular, reproducible framework for secure, audio-informed authentication applicable to specialised security environments.

Contents

1 Introduction **1**

2 Methodology **5**
2.1 Development Methodology 5
2.1.1 Project Scope and Initial Methodology 5
2.1.2 Agile Evolution 5
2.1.3 Development Phases and Timeline Management 6
2.2 Environment Setup 8
2.2.1 Environment Configuration and Challenges 8
2.2.2 Audio Dataset Preparation 8
2.3 Requirements Gathering and Analysis 8
2.3.1 Initial Requirements 8
2.3.2 Review of Related Systems 9
2.3.3 Use Case Development 9
2.4 System Architecture 10
2.5 Implementation Methodology 10
2.5.1 Component-Based Implementation 10
2.5.2 Secure Credential Handling 11
2.5.3 User Interface and Interaction Logic 11
2.5.4 Security Testing and Evaluation Tools 11
2.6 Evaluation Methodology 12

3 Technology Review **14**
3.1 Audio Processing Technologies 14
3.1.1 Digital Audio Fundamentals 14
3.1.2 Feature Extraction and Stability 15
3.2 AI/ML Technologies for Password Generation 16
3.2.1 Neural Approaches in Audio-Based Systems 16
3.2.2 Generative AI for Password Creation 17
3.2.3 Controlling Entropy in Generated Passwords 17
3.2.4 Security Evaluation of Generated Outputs 17
3.3 Cryptography and Hashing Techniques 18

3.3.1 Cryptographic Hash Functions 18
3.3.2 Symmetric Encryption 19
3.3.3 Password Security Fundamentals 20
3.3.4 Modern Password Storage Best Practices 21
3.4 Database Technologies for Secure Storage 22
3.4.1 Relational vs. NoSQL Approaches 22
3.4.2 SQLite for Local Security Applications 23
3.4.3 Database Access Patterns 24
3.5 AI in Cybersecurity: Benefits and Dangers 25
3.5.1 Applications in Detection and Assessment 25
3.5.2 Benefits in Pattern Analysis and Speed 25
3.5.3 Risks and Limitations 26
3.5.4 AI Governance Considerations 27
3.6 Integration of Technologies in Authentication Systems 27
3.6.1 Multi-factor Authentication Considerations 27
3.6.2 Human Factors in Security 28
3.7 Conclusion 29

4 System Design 30
4.1 System Architecture Overview 30
4.1.1 Architectural Principles 30
4.1.2 System Components and Interactions 31
4.1.3 Technology Stack 33
4.2 Component Design and Implementation 33
4.2.1 Audio Processing Module 33
4.2.2 Feature Extraction Module 33
4.2.3 Hash Generation Module 34
4.2.4 Password Generation Module 35
4.2.5 Storage and Retrieval Module 35
4.3 Database Design 36
4.3.1 Schema Design 36
4.3.2 Data Security Implementation 36
4.3.3 Query Patterns 37
4.4 User Interface Design 37
4.4.1 Interface Architecture 37
4.4.2 User Workflows 38
4.4.3 Usability Considerations 39
4.5 Security Implementation Details 39
4.5.1 Authentication Security 39
4.5.2 Data Protection 40
4.5.3 Communication Security 40

4.5.4 System Hardening 40
4.6 Integration and Testing Architecture 41
4.6.1 Integration Approach 41
4.6.2 Testing Infrastructure 41

5 System Evaluation 42
5.1 Evaluation Against Original Objectives 42
5.1.1 Objective Review 42
5.1.2 Feature Extraction Reliability 43
5.1.3 Password Generation Security 43
5.1.4 System Usability 44
5.1.5 Test Coverage Overview 45
5.2 Security Evaluation 45
5.2.1 Entropy and Complexity 46
5.2.2 AI Prediction Resistance 47
5.2.3 Brute Force Resistance 49
5.2.4 Pattern Analysis 50
5.2.5 Audio Feature Correlations 51
5.3 Performance Metrics 51
5.3.1 Processing Efficiency 52
5.3.2 Scalability Assessment 52
5.3.3 Reliability Metrics 52
5.4 System Limitations 53
5.4.1 Technical Limitations 53
5.4.2 Application Context 53
5.5 Comparison with Other Authentication Methods 54
5.5.1 Traditional Password Systems 54
5.5.2 Biometric Authentication Methods 54
5.5.3 Comparison with Voice-Based Password Generation 55
5.6 Future Work 58
5.6.1 Performance Optimisation 58
5.6.2 Pattern Reduction 58
5.6.3 Secure Audio File Handling 58
5.6.4 Expanded Application Contexts 59

6 Conclusion 60

Appendices 64
Appendix A: Feature Extraction and Hash Generation 64
Appendix B: AI-Based Password Generation 65
Appendix C: Cryptographic Pipeline 66

Appendix D: Fine-Tuned Model and Security Testing 67
Appendix E: Security Metrics and Analysis Tools 68
Appendix F: Audio-Password Correlation Analysis 69
Appendix G: GitHub Repository and Installation Instructions 70

References **73**

List of Figures

2.1 Initial sprint schedule created in Jira with rigid sprint-based deadlines. 7
2.2 Updated project schedule reflecting overlapping phases after GitHub transition. 7
2.3 Moderation rejection encountered during OpenAI fine-tuning due to flagged terms such as "password" and "cracking". 12

3.1 Audio Feature Extraction Pipeline with Feature Descriptions 15
3.2 Structured Prompt for GPT-4o Password Generation Based on Audio Features 16
3.3 Cryptographic Pipeline for Password Generation and Authentication 20

4.1 System Data Flow from Audio Input to Secure Storage 32
4.2 System architecture and component responsibilities. Green: Audio password system (Cormac), Orange: Vocal input system (James), Blue: Shared backend and analysis components. 32
4.3 Main user interface showing login, account creation, and password testing options 38

5.1 Comparison of character class usage between audio-generated passwords and dictionary-based passwords. 44
5.2 Password entropy comparison between audio-generated passwords (original and adjusted) and dictionary-based passwords. 46
5.3 NIST-based password complexity score comparison between audio-generated passwords and dictionary-based passwords. 47
5.4 Comparison of character class distributions between original audio-generated passwords and AI-predicted passwords. 48
5.5 Distribution of edit distances between AI-predicted passwords and true passwords, showing high divergence and unpredictability..... 49
5.6 Estimated time required to brute-force the generated passwords under different attack models. 49
5.7 Heatmap showing correlations between extracted audio features and generated password properties. Stronger relationships support feature-informed password design. 50

5.8 Average execution time breakdown across different stages of the authentication pipeline, highlighting feature extraction as the primary bottleneck. 51
5.9 Comparison of the audio-based password system against traditional biometric authentication methods across security, spoofing resistance, user convenience, and privacy. 55

List of Tables

4.1 Feature importance showing correlation with password properties and stability ratings 34
4.2 Database schema with relationships and indexing strategy 36

5.1 Pytest coverage summary across both development directories 45
5.2 Sample of AI fine-tuned password candidates generated from extracted audio features, with corresponding edit distances from the true password 48
5.3 Comparison of Voice-Based and Audio-Based Password Generation Systems 57

Chapter 1

Introduction

In today's digital landscape, authentication systems form the first line of defense for protecting sensitive information and services. Traditional authentication methods—primarily passwords—suffer from well-documented limitations. Users create weak, predictable passwords for memorability, reuse them across services, and frequently forget them [1]. This creates a fundamental tension between security and usability that has persisted for decades. Meanwhile, biometric authentication offers convenience but raises serious privacy concerns and presents the irrevocable nature of biological characteristics [2].

This project explores an alternative approach: leveraging audio files as a basis for authentication. By extracting stable features from audio and using them to generate and retrieve strong passwords, the system aims to balance security requirements with practical usability. The audio-based approach occupies a unique position in the authentication landscape—it eliminates the burden of password memorisation while avoiding the privacy implications of storing biological identifiers. It also aligns with the privacy-preserving principles set out in ISO/IEC 24745 for biometric information protection [3].

The significance of this work lies in addressing authentication challenges for high-security domains such as financial services, healthcare systems, and critical infrastructure access. These environments prioritise security over authentication speed and frequency, making them ideal candidates for an approach that offers enhanced protection at the cost of slightly longer processing times. By binding authentication to digital artifacts that users already possess (music, recordings, or other audio content), the system aims to improve security without introducing unfamiliar hardware requirements or complex user behaviours. The approach is consistent with NIST SP 800-63B guidance on authenticator assurance levels for high-risk environments [4].

As attack sophistication against traditional authentication systems increases, there is growing need for alternatives that resist both automated and targeted

approaches [5]. The integration of artificial intelligence in this project serves dual purposes: enhancing the generation of secure credentials and thoroughly evaluating resistance against advanced prediction techniques [6]. This exploration of AI's role in both strengthening and testing authentication mechanisms reflects the evolving relationship between AI and security systems. Recent research shows that state-of-the-art neural techniques can recover more than 35 % of the RockYou2023 password dataset after only 10^9 guesses—orders of magnitude faster than traditional cracking tools—according to SOPGesGPT results [7], underscoring the urgency for alternative factors.

The project was guided by four primary objectives:

1. **Stable Feature Extraction:** Develop a system capable of extracting consistent, reproducible features from audio files, ensuring that identical audio inputs reliably generate identical cryptographic identifiers across multiple processing instances.

2. **Secure Password Generation:** Create a mechanism that produces cryptographically strong passwords based on audio features, maintaining high entropy and resistance to prediction while preserving a meaningful relationship with the source audio characteristics.

3. **Secure Storage and Retrieval:** Implement encrypted storage of credentials with proper key derivation, ensuring that authentication requires both knowledge (username) and possession (specific audio file) factors.

4. **Comprehensive Security Evaluation:** Develop and apply rigorous testing methodologies to assess the system's resistance to various attack vectors, including brute force attempts, pattern analysis, and AI-based prediction.

Recent advances in robust audio fingerprinting [8] and contrastive-learning embeddings [9] informed the feature-extraction pipeline adopted here.

Success metrics were established for each objective to guide both development and evaluation. Feature extraction was considered successful if it produced identical hashes for the same audio file with at least 95% consistency. Password generation needed to achieve minimum entropy of 60 bits, include all character classes, and resist prediction attempts. Work by Raghunath *et al.* demonstrates that audio-derived randomness can meet modern entropy targets for symmetric-key systems [10]. Storage mechanisms required proper encryption, secure key derivation, and protection against exposure of plaintext credentials. The testing framework needed to provide quantitative security metrics across multiple attack vectors.

The project focuses specifically on using pre-recorded audio files as an authentication factor, rather than attempting to address all authentication challenges.

It targets controlled environments where authentication frequency is limited and security requirements are high, explicitly acknowledging that the approach is less suitable for rapid, frequent authentication scenarios like consumer web services.

The implementation is intended as a proof of concept and research exploration rather than a production-ready system. While security and functionality are prioritised, some aspects that would be necessary for commercial deployment—such as comprehensive user management, account recovery mechanisms, and cross-platform support—remain outside the project scope.

The security evaluation focuses on the technical properties of the system rather than organisational security practices or deployment considerations. It assumes that basic security hygiene (e.g., secure operating environments, protection against physical access) is maintained through standard practices outside the authentication system itself.

This dissertation is organised into six chapters:

- **Chapter 1: Introduction** establishes the context, objectives, and scope of the project, providing the foundation for understanding the work's relevance and approach.
- **Chapter 2: Methodology** describes the development approach, environment setup, and organisational aspects of the project, explaining how the work was structured and conducted.
- **Chapter 3: Technology Review** examines the technical foundation for the project, including audio processing techniques, cryptographic methods, artificial intelligence applications, and database technologies relevant to authentication systems.
- **Chapter 4: System Design** details the architecture and implementation of the audio-based authentication system, explaining component interactions, data flows, and security mechanisms.
- **Chapter 5: System Evaluation** assesses the implementation against the original objectives, presenting security analysis, performance metrics, and comparative evaluation with alternative approaches.
- **Chapter 6: Conclusion** summarises key findings, identifies contributions, and reflects on potential implications and future directions for audio-based authentication.

The full integrated project codebase is available at `https://github.com/JamesDoonan1/sound-to-security` and includes the combined authentication systems and unified user interface.

This dissertation focuses specifically on the audio-based password generation system, available independently at `https://github.com/JamesDoonan1/sound-to-security/tree/feature/audio-passwords`, which includes the following key components:

- `PasswordGenerator/` - Core authentication system components including audio processing, feature extraction, hashing, password generation, and database operations
- `AI_Training/` - Specialised components for fine-tuning and testing AI models for security evaluation
- `PasswordAnalysis/` - Analysis tools for evaluating password strength, brute force resistance, and correlation between audio features and password characteristics
- `ui.py` - Graphical user interface for account creation, authentication, and security testing

The repository includes a comprehensive README file with setup instructions, dependency information, and usage examples for both authentication and testing components.

Chapter 2

Methodology

2.1 Development Methodology

2.1.1 Project Scope and Initial Methodology

The project aimed to develop an audio-based password generation system that securely derives authentication credentials from user-provided audio files. Responsibility for development was divided between two team members: the focus of this dissertation being on workflows based on pre-recorded audio files, while the complementary stream addressed real-time vocal input. Key functional objectives for the pre-recorded audio file workflow included extracting reproducible audio features, generating AI-enhanced passwords, encrypting credentials using cryptographic best practices, and ensuring consistent authentication outcomes. The system was designed to support both pre-recorded audio file workflows and real-time vocal input, enabling flexible deployment in local security applications.

Development followed an Agile-inspired Scrum approach that evolved over the project timeline [11]. With two team members focusing on complementary areas—audio file-based and real-time voice input respectively—the methodology needed to support parallel development while maintaining consistent integration.

The project codebase and documentation were maintained under version control using GitHub from the outset of development. This ensured consistent collaboration, backup, and change tracking across all project phases.

2.1.2 Agile Evolution

Initially, project management was structured using Jira, creating user stories with story points and task breakdowns for initial sprints and product backlog. This formal Agile approach helped establish clear objectives: developing audio feature extraction, creating the password generation mechanism with AI integration, im-

plementing secure storage solutions, and, finally, ensuring comprehensive testing was completed. However, as the semester progressed and competing academic commitments intensified, the rigid sprint schedules became difficult to maintain alongside other coursework.

In response to these challenges, the methodology was adapted to a more flexible GitHub-centred workflow. The transition from Jira to GitHub Issues allowed for the streamlining of tracking specific technical tasks while maintaining the visibility of progress across both development streams. This modified approach preserved the core Agile principles of incremental development and regular feedback.

2.1.3 Development Phases and Timeline Management

The development cycle throughout consisted of daily check-ins, conducted either in person or virtually, monthly in-person collaboration sessions, and fortnightly meetings with the project supervisor, who effectively served as the Product Owner, and facilitated sprint reviews to ensure consistent progress was being achieved. Together, this ensured that:

1. Features developed incrementally with MVPs (Minimum Viable Products),
2. The password generation system was refined through iterations,
3. Testing was integrated throughout development,
4. Adaptation was based on integration outcomes.

The six main development phases were initially outlined in the original Gantt chart (as shown in Figure 2.1). However, after switching to GitHub, the schedule was revised to support overlapping task progress, as shown in the updated Gantt chart (as shown in Figure 2.2). While the original plan experienced delays, the revised timeline allowed the remaining phases to complete on time. This adaptation reflected core Agile risk management practices, responding to emerging challenges through flexible scheduling and reprioritisation.

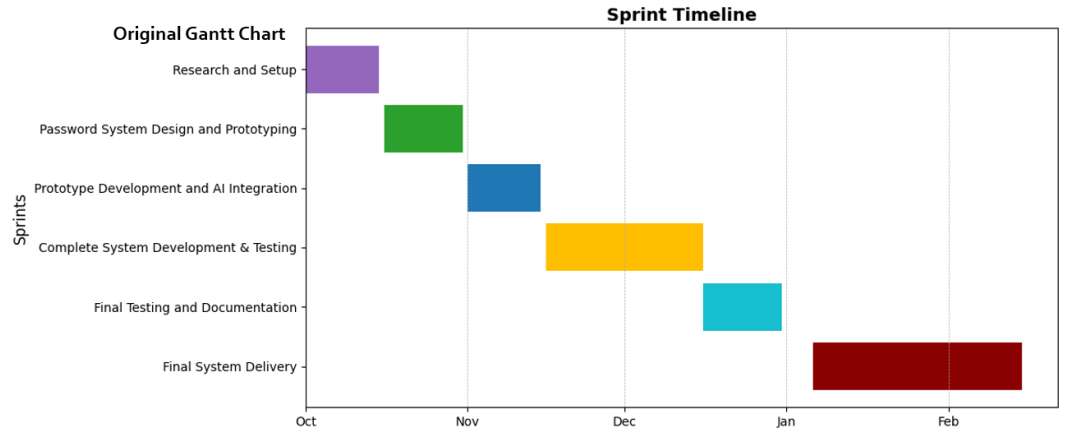

Figure 2.1: Initial sprint schedule created in Jira with rigid sprint-based deadlines.

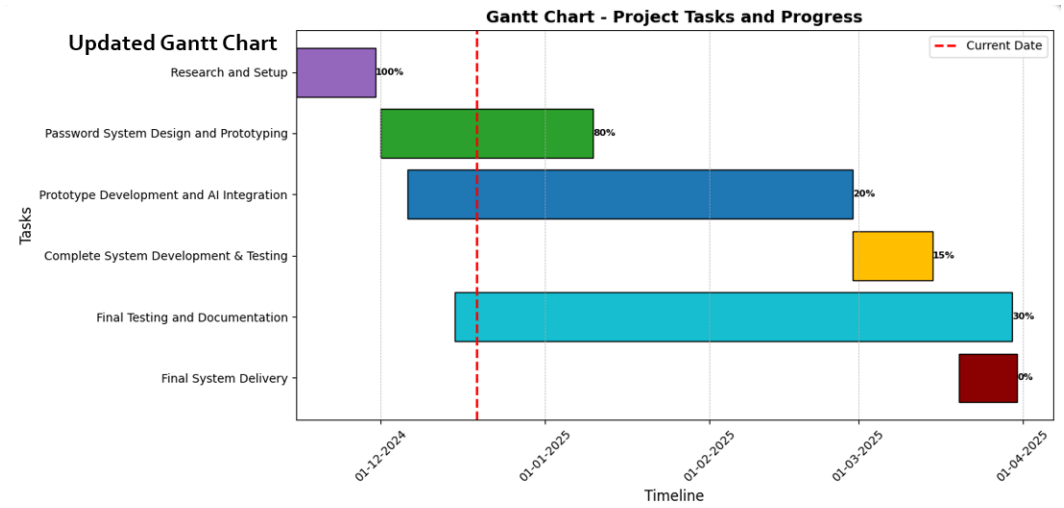

Figure 2.2: Updated project schedule reflecting overlapping phases after GitHub transition.

The adopted Agile-based approach was validated through regular supervisor feedback, continuous integration testing, and modular development, ensuring that incremental goals were consistently met and that project objectives remained aligned with academic and technical standards.

2.2 Environment Setup

2.2.1 Environment Configuration and Challenges

The development environment was established using Oracle VirtualBox [12] while Ubuntu was chosen due to its compatibility with key open-source libraries required for audio processing and cryptographic operations. This then was configured with Python and Visual Studio Code. This virtualised approach ensured consistent dependency management while isolating the development environment from the host system.

Initial setup took approximately two weeks, with the Ubuntu installation especially posing significant challenges, requiring extensive trial and error to achieve a stable configuration. Setting up a system-wide Python environment also proved difficult, and RAM allocation problems between the VM and the host machine caused random crashes. These technical issues contributed heavily to the setup time but were eventually resolved.

2.2.2 Audio Dataset Preparation

To support audio processing development and testing, a shared folder accessible from both the host machine and the VM was created and populated with 1,000 diverse audio files. This dataset provided a robust foundation for testing feature extraction stability and training the AI password generation components. Ensuring easy access between environments facilitated efficient experimentation and iteration throughout the project.

2.3 Requirements Gathering and Analysis

2.3.1 Initial Requirements

The initial requirements for the audio-based password generation system were identified through a comprehensive process involving literature review, technical feasibility assessment, and security needs analysis. This phase began with an examination of existing authentication mechanisms and their limitations, particularly focusing on the balance between security and usability. Several key requirements emerged from this analysis: the system needed to extract meaningful features from audio files, generate cryptographically strong passwords and hashes based on these features, store them securely, and consistently reproduce the same hashes when presented with the same audio input.

2.3.2 Review of Related Systems

Research into existing audio-based security systems revealed limited implementation in authentication contexts. Most prior work focused on audio fingerprinting for content identification or voice biometrics, rather than using general audio characteristics for cryptographic purposes. This gap represented an opportunity for innovation. Systems utilising audio fingerprinting for content identification and voice biometric authentication methods were examined to understand potential audio feature extraction techniques, while established password management systems were analysed to identify best practices for security.

2.3.3 Use Case Development

The project developed several use cases to guide development. Primary scenarios included:

1. New user registration using a username and choosing an audio file to generate a secure password,

2. User authentication using the same username and audio file to retrieve the password,

3. Security testing to evaluate resistance against various attack vectors,

4. Handling incorrect audio inputs gracefully by rejecting authentication attempts when provided audio does not match the registered data,

5. Secure password regeneration to ensure consistent outputs when users reupload their original audio input.

These use cases were instrumental in guiding system design and development priorities. Emphasising secure and deterministic hash generation during registration and authentication scenarios ensured functional consistency, a key requirement for usability. Security-focused use cases provided proactive identification of potential vulnerabilities, allowing early mitigation of risks such as brute-force or dictionary attacks. Furthermore, clearly defining error-handling requirements for incorrect audio inputs reinforced the importance of user feedback mechanisms, helping maintain the system's reliability and user trust. The secure regeneration scenario was particularly critical, highlighting the importance of stable and repeatable audio feature extraction methods to ensure that hashes could be reliably reproduced without compromising security or convenience.

2.4 System Architecture

The system architecture was designed with modular components to promote reproducibility, maintainability, and security. This modularity ensured that individual components could be developed, tested, and improved independently without introducing instability into the system as a whole. The architecture included distinct modules for audio feature extraction, hashing, AI-driven password generation, encryption, and database management. These modules were selected based on literature review findings indicating that separation of concerns improves system security and maintainability. Consistency across modules was critical, particularly for ensuring stable feature extraction and reproducible authentication workflows. During development, initial prototypes highlighted the need for strict module boundaries and standardised interfaces between components to manage complexity. A detailed explanation of each module, along with system diagrams, is provided in the System Design chapter.

2.5 Implementation Methodology

2.5.1 Component-Based Implementation

The audio-based password generation system was implemented using a modular development strategy aligned with the component-based architecture outlined during system design. Python was selected as the primary programming language due to its strong support for audio processing, AI integration, and rapid prototyping.

The audio feature extraction module leveraged the Librosa library [13] to process audio files, focusing on a diverse set of characteristics such as MFCCs, spectral centroid, tempo, beats, harmonic and percussive separation, zero-crossing rate, and chroma features. Special handling was implemented for edge cases, including very short or silent clips, to ensure robust feature extraction across varied input types.

The feature arrays were then normalised and formatted for reproducibility. The hashing module generated deterministic MD5 hashes from these processed features, ensuring that identical audio files consistently yielded the same identifier. A large focus was made to rounding precision and feature ordering to eliminate non-deterministic behaviour during hash generation.

The AI-powered password generation component used the OpenAI GPT-4o API [14]. Structured prompting based on extracted audio features informed the password construction process, mapping characteristics like brightness and tempo into password attributes. API interactions were wrapped in error-handling and rate-limiting logic to maintain system stability during development and testing phases.

2.5.2 Secure Credential Handling

To safeguard credentials, the encryption and storage modules followed strict security practices. Passwords generated during account creation were immediately encrypted using Fernet symmetric encryption [15], with keys derived through PBKDF2-HMAC-SHA256 [16] based on the deterministic audio hash and a cryptographically secure random salt.

User records in the SQLite database were stored using parameterised queries to prevent SQL injection attacks. Each record associated a username, a reproducible audio hash, a cryptographic salt, and the encrypted password. During login, the stored salt enabled accurate key re-derivation without requiring additional user input beyond the username and original audio input. If no match was found, users were prompted to register a new account.

2.5.3 User Interface and Interaction Logic

The user interface was built using Tkinter, providing a lightweight and accessible front-end for account creation, login, and testing operations. The UI maintained clear separation of concerns by interacting directly with backend modules to handle user inputs, trigger feature extraction and password retrieval, and display success or error messages appropriately.

The interface design prioritised simplicity to minimise user friction, ensuring that tasks such as selecting an audio file, entering a username, and authenticating could be completed intuitively without compromising security workflows.

2.5.4 Security Testing and Evaluation Tools

A separate AI fine-tuning workflow was developed exclusively for adversarial testing purposes. This module converted extracted audio features and associated passwords into JSONL format for fine-tuning custom GPT models via the OpenAI API. Attack simulations against the password generation system allowed for rigorous evaluation without impacting production workflows.

During fine-tuning setup, a significant issue was encountered with OpenAI's moderation system. Attempts to train the model using examples containing phrases such as "password" and "cracking" were automatically rejected, preventing the fine-tuning process from proceeding. This unexpected obstacle required redesigning the training prompts to align with moderation policies without compromising the adversarial testing objectives.

The prompts were revised to request the generation of "identifiers" and "access codes" instead of directly referencing passwords or password cracking. This

adjustment preserved the intended relationship between audio features and generated credentials while ensuring that moderation policies were satisfied. After these changes, fine-tuning was successfully completed without further rejection issues.

Evidence of the moderation rejection encountered during fine-tuning is shown in Figure 2.3.

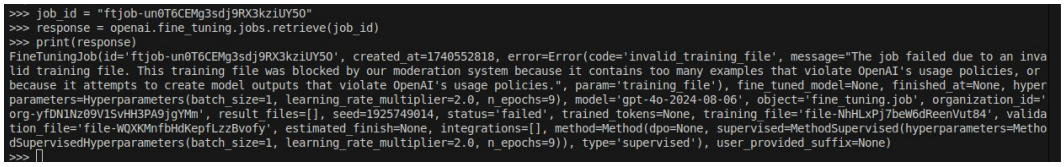

Figure 2.3: Moderation rejection encountered during OpenAI fine-tuning due to flagged terms such as "password" and "cracking".

Additionally, a suite of stand-alone evaluation tools was implemented as independent Python scripts. These tools measured execution time across varying input sizes, assessed password entropy and strength, performed brute force simulations, and conducted feature correlation analysis. Results from these tools informed iterative refinements throughout development, ensuring continuous validation of system security and performance.

Testing was conducted using the pytest [17] framework to ensure modular validation across all system components. Unit tests were developed for each module, including audio feature extraction, hashing pipelines, AI password generation, encryption routines, database management, and user interface logic. Integration tests validated the end-to-end workflow, confirming that feature extraction correctly influenced password generation and that encrypted credentials could be reliably stored and retrieved.

Particular attention was paid to security-focused testing, evaluating cryptographic operations, input handling robustness, and database query protections against SQL injection. In total, over 100 individual tests were written and executed across the development directories to validate functionality, stability, and security. Testing was an ongoing activity integrated throughout development rather than being deferred to later stages. This multi-layered test suite maps onto the structured taxonomy of security-testing approaches proposed by Felderer *et al.*[18], ensuring coverage from unit through adversarial levels.

2.6 Evaluation Methodology

Evaluation of the audio-based password generation system was designed to systematically assess performance against project objectives. Key evaluation dimensions

included password strength (entropy analysis, character class diversity), reproducibility of feature extraction and hashing workflows, security robustness against brute-force and AI-assisted attacks, and system performance under varied input conditions.

Comparative evaluation was conducted against traditional dictionary-based password schemes to benchmark system advantages and limitations. Attack simulations, both rule-based and machine learning-driven, were designed to quantify system resilience. Execution profiling was used to ensure the system remained responsive across a range of audio file sizes.

The methodology for evaluation was designed to support quantitative benchmarking and security testing, with full results discussed separately in the System Evaluation chapter.

Chapter 3

Technology Review

This chapter reviews the key technologies used in the implementation: digital audio processing (Librosa), AI-guided password generation (OpenAI GPT-4o), secure hashing and encryption (MD5, PBKDF2, Fernet), and local storage (SQLite). Each is evaluated for conceptual suitability, implementation trade-offs, and contribution to the system's objectives.

3.1 Audio Processing Technologies

3.1.1 Digital Audio Fundamentals

Digital audio analysis begins with representing continuous sound waves as discrete numerical samples. Sampling, typically at 44.1 kHz for CD-quality audio, captures amplitude values at regular intervals. Bit depth (e.g., 16 or 24 bits) determines the precision of these values. These samples can then be transformed into the frequency domain using Fourier Transform techniques, such as the Fast Fourier Transform (FFT), to reveal spectral content critical for audio-based identification tasks.

Audio formats directly affect feature extraction quality. Lossless formats like WAV and FLAC preserve all audio data, making them suitable for security contexts requiring precise analysis. In contrast, lossy formats like MP3 introduce compression artifacts that can affect reproducibility, particularly in high-frequency components.

To ensure reliability, this project operates on pre-recorded digital audio files, avoiding environmental variables like microphone quality, ambient noise, or room acoustics. Librosa is used to standardise inputs by applying mono channel conversion, fixed resampling rates, and amplitude normalisation [13]. This preprocessing ensures consistency across inputs of varying quality or format.

3.1.2 Feature Extraction and Stability

The implemented system extracts a range of spectral and temporal features using Librosa. These include Mel-Frequency Cepstral Coefficients (MFCCs) [19], spectral centroid, spectral contrast, spectral roll-off, zero-crossing rate [20], tempo and beat count [21], chroma features [22], and harmonic–percussive separation [23]. The harmonic–percussive separation stage echoes the deterministic-plus-stochastic signal model introduced by Serra and Smith [24]. Each feature contributes a different perspective on the signal's tonal and rhythmic properties. Classic sub-band fingerprinting demonstrates that such compact feature sets can enable sub-second audio identification at scale [25]. Building on that foundation, more recent hashing pipelines remain robust even under aggressive channel distortion and codec cascades [8].

MFCCs, in particular, are stable and reproducible across processing runs and are weighted more heavily in the final feature vector. Spectral centroid and contrast also showed high stability and discriminative potential. Conversely, features like tempo and beat count were more sensitive to internal variations and thus weighted less heavily in the final hash vector.

To ensure consistent password generation from identical audio inputs, all extracted features are rounded and formatted deterministically before being passed to the hashing stage. This preserves reproducibility and avoids small internal variations influencing the output hash.

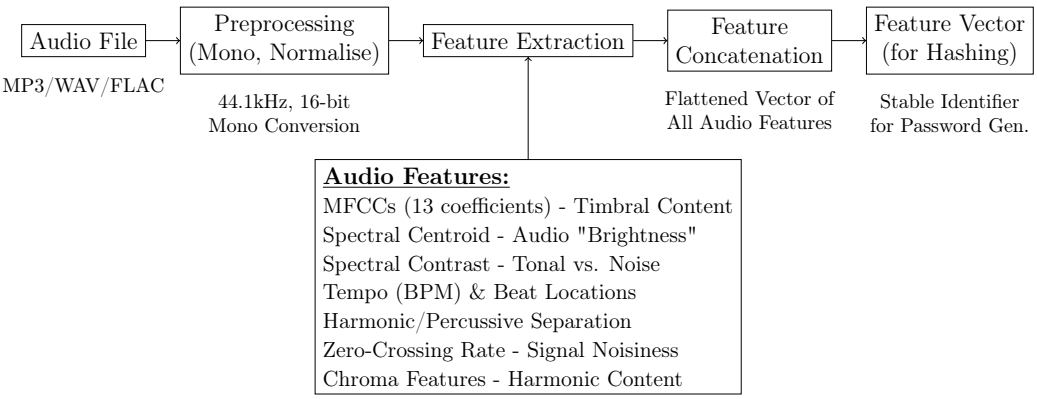

Figure 3.1: Audio Feature Extraction Pipeline with Feature Descriptions

3.2 AI/ML Technologies for Password Generation

3.2.1 Neural Approaches in Audio-Based Systems

Neural networks, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have achieved strong performance in audio classification by learning hierarchical patterns from raw audio or spectrograms, with attention-augmented CNN-RNN hybrids now represent the state of the art in musical-instrument tagging [9]. However, their reliance on large datasets and stochastic training processes limits their suitability for reproducible systems. The password generation system implemented in this project avoids these models in favour of deterministic feature extraction via Librosa to ensure consistency across authentication attempts.

```
Generate a secure password that reflects these audio
characteristics:

- Bright, high-frequency tones (use symbols like ^, *)
  [Based on spectral centroid value: {mean_spectral_centroid:
  .2f}]
- Rich harmonic profile (mix uppercase and lowercase letters)
  [Based on MFCC value: {mean_mfcc:.2f}]
- Energetic tempo ({tempo:.2f} BPM) → include more numbers
- Many beats ({beats_count}) → use more special characters

Requirements:
- Length exactly 12 characters.
- Include uppercase and lowercase letters.
- Include numbers.
- Include special characters.
- No obvious patterns.
- Must be cryptographically strong.

Return only the password with exactly 12 characters and no extra
text.
```

Figure 3.2: Structured Prompt for GPT-4o Password Generation Based on Audio Features

3.2.2 Generative AI for Password Creation

Generative language models such as OpenAI's GPT-4o are capable of producing structured text outputs with controllable variability. This project integrates GPT-4o [14] via API for password generation, using custom prompt engineering to guide the model in mapping audio feature descriptors to password elements.

Rather than training a new model, the system uses in-context learning to shape outputs. Prompt templates specify rules for length, character classes, and password formatting, while feature-derived values such as spectral centroid and beat count influence symbol use, casing, and numeric patterns. This structured prompting enables a balance between consistency and entropy without modifying model weights.

3.2.3 Controlling Entropy in Generated Passwords

Security in password generation depends on producing high-entropy outputs while ensuring that identical inputs yield identical results. Attackers now train representation-learning guessers that cut the cost of offline cracking by orders of magnitude [6], and clever "search-order" tools powered by modern language models can now churn through the leftover possibilities almost completely [7], so entropy requirements are higher than ever. To manage this trade-off, the system uses:

- **Hashing:** Stable audio features are pre-hashed using MD5 to anchor determinism.
- **Prompt conditioning:** Audio statistics influence password structure without directly encoding values.
- **Temperature and sampling controls:** Output diversity is moderated via model parameters (`temperature`, `top-k`, and `top-p`).

This architecture ensures that feature-guided passwords remain cryptographically strong and resistant to trivial inference while maintaining reproducibility.

3.2.4 Security Evaluation of Generated Outputs

The evaluation process tests both strength and resistance to model-based inference attacks. Standard metrics include:

- **Shannon entropy:** Used to assess randomness in password character distributions, providing a measure of unpredictability.

- **Pattern analysis:** Includes frequency analysis of bigrams, trigrams, and positional bias to detect structural predictability in generated passwords.
- **Adversarial inference testing:** Performed using a separately fine-tuned model trained on audio-feature-to-password mappings to simulate attacker attempts at prediction.

The results showed that despite deterministic inputs, password outputs retained high entropy and resisted adversarial model inference, validating the design's security under realistic threat models.

3.3 Cryptography and Hashing Techniques

3.3.1 Cryptographic Hash Functions

Cryptographic hash functions serve as fundamental components in modern security systems, transforming arbitrary input data into fixed-length output values. These functions exhibit several critical properties that make them suitable for security applications. Collision resistance ensures the computational infeasibility of finding two different inputs that produce the same hash output. Pre-image resistance prevents the derivation of input data from its hash value, effectively making the function one-way. Second pre-image resistance guarantees that, given an input and its hash, finding another input with the same hash remains computationally infeasible. Additionally, the avalanche effect ensures that minimal changes to the input result in substantially different hash outputs, with changes to approximately half the output bits.

Common hash algorithms have evolved significantly over time, addressing emerging vulnerabilities. MD5, once widely used, generates 128-bit hash values but is now considered cryptographically broken since 2008 [26] due to demonstrated collision vulnerabilities. SHA-1, which produces 160-bit digests, has similarly been deprecated following practical collision attacks. SHA-256, part of the SHA-2 family, generates 256-bit outputs and remains widely used in security-critical applications. SHA-3, based on the Keccak algorithm, was standardised in 2015 to provide an alternative approach to the SHA-2 family, offering resistance against attacks that might compromise SHA-2 algorithms. Each algorithm represents different trade-offs between security, performance, and output size.

In password systems, hash functions fulfil multiple critical roles. They enable secure storage by converting plaintext passwords into hash values, ensuring that even system administrators cannot access original passwords. During authentication, the system hashes the provided password and compares it with the stored hash, rather than comparing plaintext values. Hash functions also facilitate key

derivation, converting passwords or other input data into suitable cryptographic keys. In the context of audio-based password generation, hash functions can stabilise variable audio features, ensuring that minor variations in extraction produce consistent outputs, thus enhancing system reliability while maintaining security properties.

While MD5 is considered cryptographically broken for security-critical applications, the implemented system uses it in a context where its vulnerabilities do not compromise overall security. Specifically, MD5 serves as a deterministic mapping function to transform variable audio features into a consistent identifier, not as a security barrier itself. Since this hash is immediately processed through PBKDF2-HMAC-SHA256 with a unique salt before being used for encryption, the cryptographic weaknesses of MD5 do not expose the system to practical attacks. The primary security requirement at this stage is consistency rather than collision or preimage resistance, making MD5's performance advantages valuable despite its deprecated status in security-critical contexts.

3.3.2 Symmetric Encryption

Key derivation functions (KDFs) transform input material, such as passwords or biometric data, into cryptographically strong keys suitable for encryption algorithms. Unlike basic hash functions, KDFs incorporate security enhancements specifically designed to resist attacks targeting key generation. PBKDF2 (Password-Based Key Derivation Function 2) [16] applies a pseudorandom function, typically HMAC-SHA-1 or HMAC-SHA-256, to the input material along with a salt value, repeating this process many times to increase computational requirements. Scrypt extends this approach by adding memory-intensive operations, making hardware-accelerated attacks more expensive. HKDF (HMAC-based Key Derivation Function) extracts randomness from input keying material and expands it to multiple keys, particularly useful when input material already has sufficient entropy.

Block ciphers encrypt fixed-size blocks of data (typically 128 or 256 bits) using a cryptographic key. The Advanced Encryption Standard (AES) represents the most widely deployed block cipher, operating on 128-bit blocks with key sizes of 128, 192, or 256 bits, and is standardised in FIPS-197 [27]. Block ciphers require a mode of operation to handle messages longer than the block size. Cipher Block Chaining (CBC) incorporates feedback from previous blocks, while Galois/Counter Mode (GCM) provides both confidentiality and authentication. In contrast, stream ciphers generate a pseudorandom keystream that is combined with plaintext data, typically through XOR operations. ChaCha20, a modern stream cipher, offers strong security with high performance, particularly on platforms without AES hardware acceleration [28].

Encryption standards continue to evolve in response to emerging threats and

requirements. AES remains the predominant encryption standard worldwide, benefiting from extensive analysis and hardware acceleration in modern processors. ChaCha20-Poly1305, a relatively newer authenticated encryption combination, pairs the ChaCha20 stream cipher with the Poly1305 message authentication code.

Transport Layer Security (TLS) 1.3, standardised in 2018, incorporates both these algorithms, recognising their security and performance advantages [29]. These standards provide established frameworks for implementing strong encryption, with implementation details often more critical to security than algorithm selection among secure options. The implemented password system utilises Fernet symmetric encryption, a high-level recipe based on AES in CBC mode with PKCS7 padding, HMAC using SHA256 for authentication, and secure random IVs [15]. Key derivation is performed using PBKDF2-HMAC-SHA256 with a randomly generated salt unique to each account, applying 100,000 iterations to transform the audio hash into a secure encryption key. This approach follows current best practices for symmetric encryption, ensuring that the stored passwords remain protected even if the database is compromised, while enabling consistent decryption when the same audio file is presented during authentication.

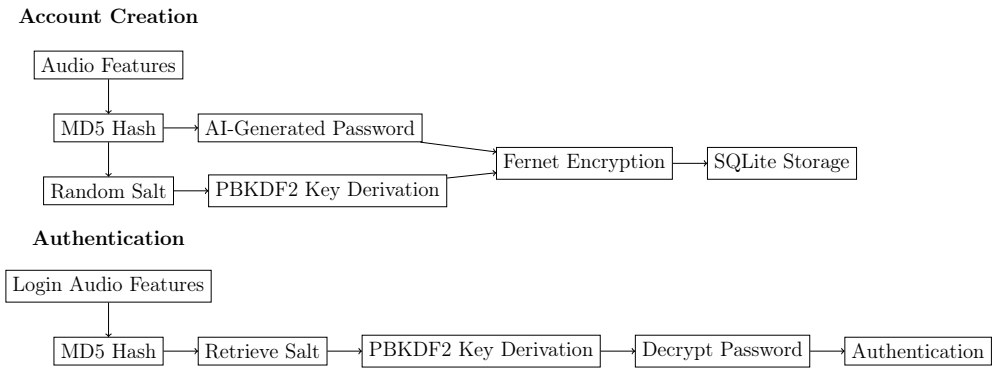

Figure 3.3: Cryptographic Pipeline for Password Generation and Authentication

The integration of these cryptographic processes into the full authentication pipeline is illustrated in Figure 3.3 and discussed in greater detail in the System Design chapter.

3.3.3 Password Security Fundamentals

Entropy in password security measures the level of unpredictability, expressed in bits. Each additional bit doubles the number of possible combinations, increasing the difficulty of a brute-force attack. A password with n bits of entropy typically requires 2^{n-1} guesses to crack on average. The size of the character set has a

significant impact on entropy. For example, a password using only lowercase letters (26 characters) has less entropy than one that includes uppercase letters, digits, and symbols (up to 95 characters). Password length exponentially affects entropy, making it the most influential factor in password strength. For a character set of size C and a password length L, the maximum theoretical entropy is calculated as $L \times \log_2(C)$ bits. However, actual entropy may be lower in practice due to uneven character distribution or predictable patterns.

Attack methods against password systems have grown increasingly sophisticated. Rainbow tables precompute hash chains for password candidates, trading storage for computational efficiency during attacks. Dictionary attacks leverage collections of common passwords and words, often incorporating simple transformations like character substitutions (e.g., "a" to "@"). Rule-based attacks apply systematic modifications to dictionary words according to observed password creation patterns. Statistical attacks leverage knowledge of character distribution and common patterns in human-generated passwords. Hybrid approaches combine these methods, starting with likely candidates before progressing to more comprehensive searches.

Defensive techniques such as salting and key stretching significantly enhance password security. Salting incorporates a random value (the salt) into the hashing process, ensuring that identical passwords produce different hash values. This approach prevents the use of precomputed tables and forces attackers to target each password individually. The salt value is typically stored alongside the hash, as its secrecy is not required for security. Key stretching techniques like iterative hashing increase the computational cost of password verification, applying the hash function repeatedly (typically thousands to millions of times). While minimally affecting legitimate authentication processes, this approach substantially increases the resources required for brute force attacks.

3.3.4 Modern Password Storage Best Practices

Password-Based Key Derivation Functions (PBKDFs) specifically address the requirements of secure password storage and key generation. PBKDF2, standardised in PKCS #5, applies a pseudorandom function repeatedly to the password combined with a salt. The iteration count parameter allows calibration of computational cost as hardware capabilities advance. Argon2, the winner of the Password Hashing Competition in 2015, addresses limitations in earlier functions by incorporating memory-hard computation, resistance to time-memory trade-offs, and defence against side-channel attacks [30]. Argon2 offers three variants: Argon2d (maximising resistance against GPU attacks), Argon2i (focusing on side-channel resistance), and Argon2id (combining properties of both).

Memory-hard functions represent a significant advancement in password hash-

ing by requiring substantial amounts of memory to compute, thereby limiting the effectiveness of specialised hardware attacks. These functions ensure that parallelisation provides minimal advantage and that memory requirements cannot be circumvented without significantly increasing computation time. Scrypt pioneered this approach by incorporating memory-intensive operations alongside iterative hashing. Argon2 refined these techniques, allowing precise configuration of memory usage, computation time, and parallelism based on system capabilities and security requirements. Memory-hardness significantly increases the cost of large-scale password cracking attempts, particularly against hardware-accelerated attacks using GPUs or ASICs.

Current standards and recommendations provide frameworks for implementing secure password systems. NIST Special Publication 800-63B establishes comprehensive guidelines for authentication and lifecycle management, recommending adaptive hashing with at least 10,000 iterations for PBKDF2 or memory-hard functions [4]. The Open Web Application Security Project (OWASP) provides practical implementation guidance, advocating for Argon2id with parameters calibrated to consume at least 15MB of memory and 2 seconds of processing time on server hardware. These recommendations emphasise the importance of algorithm selection, proper parameter calibration, and regular reevaluation as computational capabilities advance. Additionally, both sources stress the importance of implementing secure credential recovery mechanisms and avoiding composition rules that might actually reduce password entropy.

3.4 Database Technologies for Secure Storage

3.4.1 Relational vs. NoSQL Approaches

Database technology selection significantly impacts security architecture, with relational and NoSQL databases offering different security models. Relational databases (RDBMS) like MySQL, PostgreSQL, and SQL Server provide robust, schema-enforced data integrity and comprehensive access control mechanisms. Their structured approach enables precise permissions at table, column, and row levels, facilitating principle of least privilege implementation. ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures data integrity during transactions, preventing partial updates that could compromise security state. Conversely, NoSQL databases such as MongoDB, Cassandra, and Redis prioritise flexibility and scalability, often with simplified security models. Document stores typically implement collection-level permissions rather than field-level controls, potentially complicating fine-grained access restrictions.

The security implications of these architectural differences extend beyond per-

mission models. Relational databases typically offer mature auditing capabilities, logging access patterns and schema changes for security monitoring. NoSQL solutions have historically provided less comprehensive audit trails, though this gap has narrowed in recent versions. Authentication mechanisms also differ, with relational databases commonly supporting integration with enterprise directory services, while NoSQL databases have traditionally emphasised lightweight authentication. For multi-tenant applications, relational databases often provide stronger isolation guarantees, critical for preventing data leakage between users or organisations.

Query patterns for password verification reflect fundamental differences between database paradigms. In relational databases, verification typically involves prepared statements with parameterised queries that select the stored password hash based on the username or identifier. This approach minimises data exposure by retrieving only the specific hash needed for verification. NoSQL approaches often retrieve entire user documents, which may contain additional sensitive information beyond authentication data. Both models should implement index-based lookups to prevent full table/collection scans that could impact performance under load—a potential denial of service vector. Query execution time should be carefully managed to prevent timing attacks that might reveal the existence of specific usernames.

3.4.2 SQLite for Local Security Applications

SQLite offers unique advantages for local security applications, functioning as a serverless, zero-configuration database engine that operates within the application process. This embedded architecture eliminates network exposure risks inherent in client-server database systems, reducing the attack surface. SQLite implements ACID properties through its journal mechanism while maintaining a minimal footprint (approximately 600KB), making it suitable for resource-constrained environments. Its self-contained design requires no separate installation or administration, simplifying deployment and reducing potential misconfiguration vulnerabilities. However, SQLite's concurrent access model uses file-level locking rather than row-level locking, potentially limiting throughput under heavy simultaneous write operations.

Security considerations for local databases centre on file system protections and access controls. SQLite database files inherit the operating system's file permission controls, making proper system-level access restrictions essential. Default SQLite implementations lack built-in user authentication, relying entirely on file system permissions to prevent unauthorised access. This simplicity demands careful attention to directory permissions, particularly on multi-user systems where different privilege levels exist. Temporary files generated during database operations may

contain sensitive information and require appropriate protection through secure file creation practices and clean-up procedures.

Encryption options for SQLite databases address the vulnerability of data at rest. The SQLite Encryption Extension (SEE) provides transparent 256-bit AES encryption of database content, though it requires a commercial license. Open-source alternatives include SQLCipher, which implements full database encryption with minimal performance overhead while maintaining compatibility with the standard SQLite API [31]. These solutions encrypt both the database file and temporary files created during operations, protecting against direct file access attacks. Proper key management remains essential, as encryption keys must be securely stored and transmitted to the application. For maximum security, keys should derive from user authentication rather than being stored locally, preventing unauthorised access even if the device is compromised.

The project implements a straightforward SQLite database schema optimised for the authentication use case. Each record in the `hash_passwords` table contains a username, audio hash, cryptographic salt, and encrypted password, with a composite primary key of username and hash ensuring uniqueness. All database interactions use parameterised queries to prevent SQL injection attacks, with separate execution paths for record creation, authentication, and password retrieval. This implementation leverages SQLite's simplicity and file-based nature while incorporating security best practices for sensitive data storage.

3.4.3 Database Access Patterns

Secure database access begins with connection management practices that balance security and performance. Connection pooling reuses established database connections rather than creating new ones for each operation, reducing authentication frequency and potential exposure of credentials. However, pooled connections must implement appropriate isolation to prevent cross-request data leakage, particularly in multi-user environments. Credentials for database access should leverage environment variables or secure credential stores rather than embedding them in source code or configuration files, mitigating risks from code repository exposure. Connection timeout policies should terminate idle connections to minimise the window of opportunity for session hijacking attempts.

Prepared statements represent a fundamental defence against SQL injection attacks, separating SQL code from data by pre-compiling statements and binding parameters separately. This approach prevents attacker-supplied input from modifying query structure, regardless of content. Unlike string concatenation methods, prepared statements ensure that special characters in user input cannot escape their intended context. Most contemporary database APIs provide native support for parameterised queries, making implementation straightforward. For ap-

plications requiring dynamic SQL structures, query construction should leverage object-relational mapping (ORM) libraries that implement safe query building with appropriate escaping and validation.

Access control implementation should follow defense-in-depth principles, combining database-level permissions with application-level authorisation. The principle of least privilege dictates that database accounts should have minimum necessary permissions, ideally restricted to specific tables and operations required for their function. Read-only accounts should be used for queries that don't modify data, limiting the impact of potential compromise. Application-level access control should validate permissions before executing database operations, preventing unauthorised access even if database credentials are obtained. Regular permission audits should verify alignment between granted privileges and actual requirements, identifying and removing excessive permissions that could expand the impact of a security breach.

3.5 AI in Cybersecurity: Benefits and Dangers

3.5.1 Applications in Detection and Assessment

Machine-learning pipelines now dominate modern intrusion-detection architectures [5]. Anomaly detection models monitor system behaviour and identify deviations indicative of intrusions, including unknown or zero-day threats. Classification models support automated incident response by categorising alerts and applying appropriate mitigation strategies. These AI-driven systems reduce response times and improve threat coverage, particularly for rapidly evolving attack vectors.

In this project, AI capabilities are also leveraged for adversarial evaluation. A secondary GPT model is fine-tuned on paired audio-feature/password data to emulate a sophisticated attacker. By attempting to infer the password solely from the exposed features, the model produces a direct measure of the pipeline's resistance to model-inversion and feature-leakage attacks. Repeated trials yield quantitative metrics, effectively turning a generative model into an automated security auditor. This methodology reflects the model-based security-testing approach outlined by Felderer et al. [18].

3.5.2 Benefits in Pattern Analysis and Speed

AI's ability to process vast amounts of security telemetry makes it essential in identifying subtle attack patterns. Deep learning models extract abstract features from logs, traffic, and endpoint data, which helps detect stealthy, multi-stage attacks. In this system, a similar capability is leveraged for password strength assessment.

Pattern recognition techniques such as bigram/trigram analysis and entropy calculations help quantify password predictability and diversity.

Speed is another key advantage. Automated AI-based assessment enables password generation and testing workflows to run quickly and iteratively. This supports real-time evaluation of password entropy and structural consistency, which is particularly important when generation depends on user-selected audio.

3.5.3 Risks and Limitations

While AI provides substantial benefits, its misuse or limitations introduce significant risks. One notable risk is adversarial attacks, where attackers can exploit model vulnerabilities by subtly altering inputs to evade detection. In password systems, this could take the form of training a surrogate model to predict password outputs based on limited knowledge of the hashing pipeline. This potential for exploitation underlines the need for continuous monitoring and refinement of AI systems in security contexts.

Additionally, ethical concerns arise when training AI models on user-generated data. This system avoids collecting or storing raw audio data or extracted feature vectors, limiting the risk of sensitive information exposure during both training and operation. Extracted features are immediately processed into cryptographic hashes, and no original audio or feature vectors are retained, reducing the possibility of leaking identifiable information even if database contents were compromised.

Safeguards like OpenAI's moderation system are also intended to prevent misuse during fine-tuning. In this project, during the attempt to fine-tune an adversarial model for security testing, the use of terms such as "password" and "cracking" caused the training data to be flagged and rejected by the moderation system. This block was overcome by restructuring the prompts to replace these flagged terms with "identifier" and "access code." Although this adaptation allowed fine-tuning to proceed successfully, it highlights a broader limitation: AI safeguards are rule-based and can be bypassed through careful rewording of inputs. Evidence of this moderation failure and its resolution was shown previously in the Security Testing and Evaluation Tools Section, with the image in Figure 2.3 illustrating the initial refusal during training.

This experience demonstrates that while automated AI moderation provides an important first layer of defense, it is not foolproof. Security systems that rely on moderation alone must account for the fact that determined adversaries can modify inputs to evade detection, reinforcing the need for deeper security controls and human oversight in AI-driven workflows.

3.5.4 AI Governance Considerations

AI regulation in cybersecurity remains an evolving area. For this system, key governance concerns include transparency of the password generation pipeline, reproducibility of results, and resistance to model-based inference. By relying on prompt-engineered password generation rather than fully end-to-end AI, the system maintains human oversight and avoids opaque model behaviours.

The design deliberately avoids storing or exposing unprocessed audio data, and all AI components are sandboxed to prevent cross-user inference. These architectural decisions reflect security-by-design principles aligned with emerging regulatory recommendations for trustworthy and auditable AI in authentication contexts.

3.6 Integration of Technologies in Authentication Systems

3.6.1 Multi-factor Authentication Considerations

The integration of audio-based authentication within multi-factor authentication (MFA) frameworks represents a novel approach to the "something you have" category. Unlike physical tokens or mobile devices, audio files offer unique advantages in portability and duplication resistance when properly implemented. These files can be stored on various media and devices while maintaining their authenticator function through the distinctive audio features rather than the storage medium itself. This characteristic distinguishes audio authenticators from hardware tokens, which are vulnerable to physical theft, while maintaining independence from specific devices like smartphone-based authenticators.

When compared with biometric factors, audio-based authentication offers significant privacy and revocability advantages. Unlike fingerprints or face scans, an audio file can simply be swapped out if it's ever leaked. That makes sound-based logins much easier to "revoke and re-issue," solving a big problem with traditional biometrics that you can't change. Because the system stores nothing biological, it also sidesteps many of the strict privacy rules (e.g., GDPR, BIPA) that kick in when you keep actual biometric data. Standards such as ISO/IEC 24745 even spell out how these revocable, privacy-friendly templates should be handled [3].

Integration challenges with existing authentication infrastructures primarily stem from the computational requirements of audio feature extraction and the need for consistent processing environments. Variable audio capture conditions between enrolment and verification can lead to feature variations that compromise reliability. Modern authentication frameworks typically expect deterministic

verification processes with minimal computational overhead, contrasting with the resource-intensive nature of audio processing. Addressing these challenges requires careful consideration of feature stability, computational optimisation, and appropriate fallback mechanisms when audio verification cannot be completed. Standardised APIs for audio feature extraction would significantly facilitate broader adoption across authentication platforms.

The implemented audio-based password generation system offers potential integration as a supplementary authentication factor rather than a stand-alone mechanism. It effectively functions as a "something you have" factor when users select unique audio files known only to them, complementing traditional password ("something you know") approaches. The system's design ensures that identical audio inputs with different account salts produce entirely different encryption keys and stored passwords, preventing credential reuse vulnerabilities across accounts. This characteristic makes it particularly suitable for integration with existing multi-factor frameworks as an additional security layer that avoids the hardware dependencies of physical tokens while maintaining the revocability that biometric methods lack.

3.6.2 Human Factors in Security

The memorability versus security trade-off represents a central challenge in authentication system design. Traditional approaches requiring users to create and recall complex passwords typically result in counterproductive behaviours such as password reuse or insecure storage. Audio-based authentication shifts this paradigm by deriving security from recognisable audio that users can associate with specific accounts. This approach leverages episodic memory rather than semantic memory, potentially improving recall while maintaining security. Research indicates that association-based memory cues significantly enhance credential recall compared to arbitrary strings, potentially reducing dependency on password managers or written records [1].

User experience considerations significantly impact authentication system effectiveness, as security mechanisms that impede workflow face resistance or circumvention. Audio-based authentication must balance security requirements with practical usability factors such as authentication time, environmental compatibility, and accessibility. Authentication processes requiring more than a few seconds typically generate user frustration, while audio playback may be inappropriate in certain contexts such as quiet environments or public spaces. Accessibility requirements must address users with hearing impairments through alternative interfaces or equivalent security options. These considerations necessitate flexible implementation approaches that adapt to various usage contexts while maintaining consistent security properties.

Adoption challenges for novel authentication methods extend beyond technical implementation to include organisational and psychological factors. Users typically resist changing established authentication behaviours unless compelling advantages are clearly communicated. Organisational adoption requires alignment with existing security frameworks and compliance requirements, potentially necessitating formal security certifications or attestations. The unfamiliarity of audio-based authentication may initially generate scepticism regarding security efficacy, requiring clear education about the underlying security principles. Successful deployment strategies typically include phased introduction, which mirrors agile recommendations for incremental process adoption [11], comprehensive user education, and clear articulation of specific advantages over traditional approaches.

3.7 Conclusion

The technologies reviewed in this chapter form the foundation of the audio-based password generation system. Each component addresses specific challenges within the authentication pipeline. Audio processing techniques extract stable features that serve as reproducible identifiers, while cryptographic functions transform these features into secure authentication tokens. AI-based password generation bridges the gap between raw audio characteristics and human-usable passwords, enhancing security through controlled randomness without compromising reproducibility. Database technologies ensure the safe storage of credentials, protecting them against compromise while facilitating efficient retrieval during authentication workflows.

The deliberate separation between audio processing, password generation, and security validation reflects a modular design philosophy that prioritises both component independence and overall system integrity. This modularity enables independent optimisation and testing of each stage while maintaining the end-to-end security properties required for a secure authentication system.

The integration of these technologies provides a novel mechanism that improves upon traditional authentication methods by offering greater portability, revocability, and resistance to adversarial attacks. The application of these components within the system's overall architecture is detailed in the following System Design chapter.

Chapter 4

System Design

The System Design chapter documents the architectural structure, core component designs, database schema, user interface workflows, security mechanisms, and integration strategies underlying the audio-based password generation system. The design process prioritised reproducibility, security, modularity, and maintainability, ensuring that each subsystem operated independently while supporting coherent, secure workflows from audio input to credential storage.

This chapter provides a detailed technical overview of system components without evaluating performance outcomes, which are discussed separately in the System Evaluation chapter.

4.1 System Architecture Overview

4.1.1 Architectural Principles

The audio-based password generation system employs a modular design architecture with five primary layers: data acquisition, feature processing, cryptographic transformation, password generation, and storage management. Each layer encapsulates specific functionality and communicates with adjacent layers through well-defined interfaces, enhancing maintainability and security. Security-by-design principles include cryptographic non-reversibility and defense in depth [32], with a one-way transformation pathway that prevents reverse-engineering of passwords from stored data. The system implements the principle of least privilege, with each component accessing only the minimum data necessary for its function. Importantly, the system avoids storing raw audio features, maintaining only cryptographic derivatives that cannot reconstruct the original audio characteristics.

4.1.2 System Components and Interactions

The system comprises five major components that process data in a sequential pipeline:

1. Audio Processor: Handles data acquisition and preparation, supporting various formats and implementing quality control.

2. Feature Extractor: Applies signal processing to extract acoustic characteristics, generating feature vectors.

3. Hash Generator: Transforms variable-length feature vectors into fixed-length cryptographic hashes.

4. Password Generator: Creates secure, reproducible passwords through AI-driven generation.

5. Storage Manager: Handles secure persistence of hashes and encrypted passwords.

Data flows unidirectionally through these components, as shown in Figure 4.1. The Audio Processor outputs normalised audio data, the Feature Extractor produces multi-dimensional feature vectors, the Hash Generator creates deterministic cryptographic hashes, the Password Generator creates passwords guided by hashes and features, and the Storage Manager persists encrypted versions along with metadata. Interfaces between components follow consistent patterns with strict data formats and validation requirements. Error handling implements a graduated response: minor issues generate warnings, significant problems trigger fallbacks, and critical failures raise exceptions to prevent further processing.

In addition to the technical data flow diagram, a broader architectural overview is shown in Figure 4.2. This diagram illustrates how system responsibilities were divided across team members, highlighting which components were shared, individually developed, or planned but ultimately not completed. It also visualises the full system structure, including AI model integration, testing pipelines, and user interface interactions.

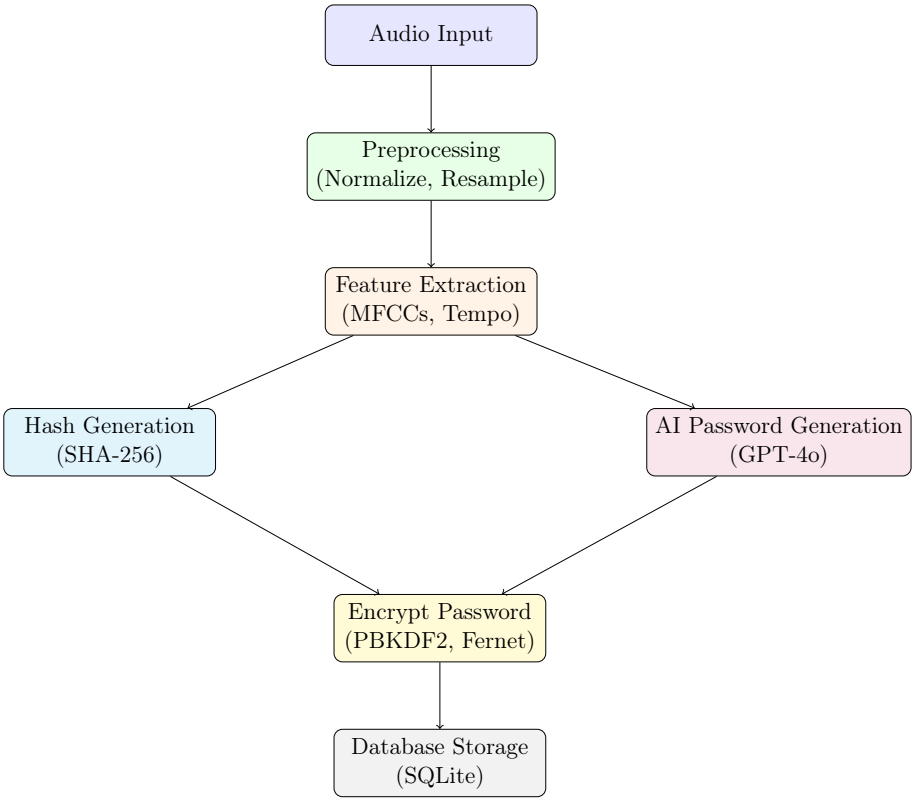

Figure 4.1: System Data Flow from Audio Input to Secure Storage

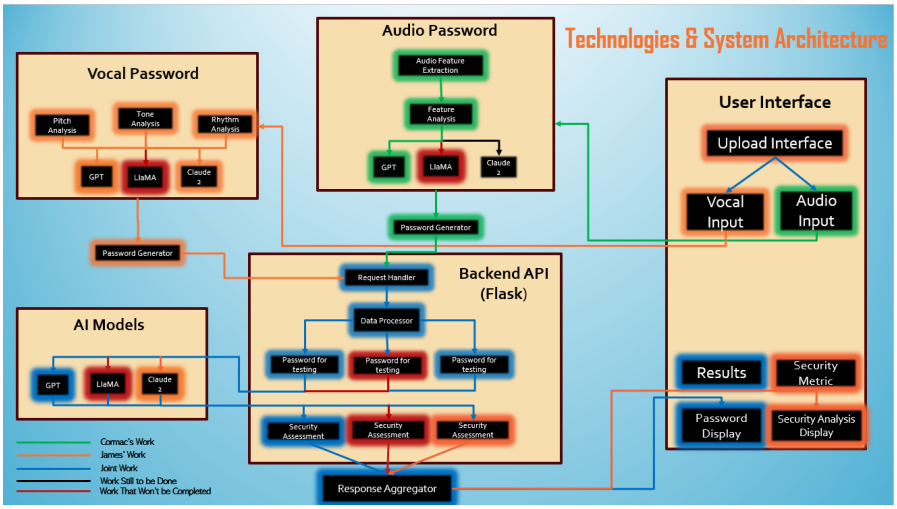

Figure 4.2: System architecture and component responsibilities. Green: Audio password system (Cormac), Orange: Vocal input system (James), Blue: Shared backend and analysis components.

4.1.3 Technology Stack

The system is implemented in Python 3.10, leveraging its scientific and cryptographic libraries. Audio processing relies on Librosa (v0.10.0) [13] for feature extraction, with NumPy providing optimised array operations. AI-driven password generation integrates with the OpenAI API targeting GPT-4o [14]. Cryptographic operations utilise the cryptography package for AES encryption and PBKDF2 for key derivation. SQLite provides a self-contained database engine that eliminates network exposure while maintaining ACID compliance. The development environment uses Oracle VirtualBox for virtualisation with Ubuntu 20.04 LTS, ensuring consistent dependency management and isolation from host system variations.

4.2 Component Design and Implementation

4.2.1 Audio Processing Module

The audio processing module serves as the system's entry point, handling the acquisition and preparation of audio data. It standardises inputs by applying mono channel conversion, resampling to 22,050 Hz, and amplitude normalisation. Signal normalisation includes peak normalisation and DC offset removal, preserving relative amplitude relationships while standardising overall signal levels [33].

Quality assessment algorithms evaluate files before processing, checking signal-to-noise ratio, duration (minimum 5 seconds), clipping, silence periods, and frequency distribution. For recordings with significant dynamic range variations, the module optionally applies compression with a soft knee characteristic. The module implements a multi-tiered approach to handling low-quality audio: minor issues generate warnings, moderate problems trigger remediation attempts, and severe corruption raises structured exceptions with remediation suggestions.

4.2.2 Feature Extraction Module

The feature extraction workflow segments normalised audio into overlapping frames (2048 samples, 512-sample hop size) with Hann windowing, then applies the Short-Time Fourier Transform to create a time-frequency representation [34]. A full implementation of the feature extraction process, including MFCCs, spectral centroid, and tempo extraction, is provided in Appendix A. Key extracted features include:

- **MFCCs:** 13 coefficients capturing timbral characteristics, particularly stable across different encodings.

- **Spectral features:** Centroid (brightness), contrast (harmonic vs. noise distribution), and rolloff.
- **Temporal features:** Zero-crossing rate, beat detection, tempo analysis, and harmonic-percussive separation.

The final feature vector combines 91 dimensions spanning timbral, spectral, temporal, and perceptual domains. Statistical summarisation (e.g., means, variances, skewness, and percentiles) reduces dimensionality while maintaining feature distribution information. Based on test results, MFCCs and tempo showed the strongest correlations with password characteristics (r=0.44 and r=0.43 respectively), confirming their importance in the feature set.

The associated stability ratings, presented in Table 4.1, quantify the consistency of each feature when extracted across repeated analyses of similar audio inputs, indicating their reliability for use in cryptographic transformations.

Audio Feature	**Correlation with Password**	**Stability Rating**
MFCC Mean	0.44 (Uppercase %)	High
Tempo	0.43 (Digit %), -0.43 (Lowercase %)	Medium
Spectral Contrast	-0.22 (Uppercase %)	Medium-High
Zero-crossing Rate	0.12 (Starts with Uppercase)	High
Spectral Centroid	0.08 (Uppercase %)	Medium-High

Table 4.1: Feature importance showing correlation with password properties and stability ratings

4.2.3 Hash Generation Module

The hash generation module applies MD5 to produce stable, reproducible hashes from audio features. These MD5 hashes are later processed through PBKDF2-HMAC-SHA256 during encryption key derivation, ensuring cryptographic strength while maintaining feature consistency. The detailed code for feature vector combination and hash computation can be found in Appendix A. The implementation converts normalised feature values to a consistent byte representation through 16-bit quantisation, arranging them in order of feature stability. Stability enhancement techniques include feature selection that prioritises the most reliable characteristics and locality-sensitive hashing [35] that preserves similarity relationships between audio segments. Hash validation verifies uniform bit distribution and entropy sufficiency before use in password generation. Security testing confirmed the effectiveness of this approach, with brute force analysis showing that even offline specialised hardware would require approximately 140 years to crack the resulting hashes.

4.2.4 Password Generation Module

AI model integration forms the core of the password generation approach, leveraging OpenAI's GPT-4o model through a structured prompting system. The feature-to-prompt mapping and the password generation request to the OpenAI API are detailed in Appendix B. Prompts follow a three-part structure: system instructions defining parameters and constraints, feature context presenting audio characteristics, and generation directives controlling output format. The mapping logic translates audio characteristics into password properties:

- Spectral centroid values influence uppercase/lowercase balance.
- Rhythmic features affect special-character distribution.
- Energy distribution impacts numeric-character frequency.
- Harmonic complexity influences overall character variety.

Password quality controls ensure minimum entropy requirements (60+ bits), representation from all character classes, absence of common dictionary patterns, and standardised 12-character length. Test results validated this approach, showing generated passwords achieve an average entropy of 78.33 bits with well-balanced character distribution: uppercase (23.4%), lowercase (37.8%), digits (12.4%), and symbols (26.4%).

4.2.5 Storage and Retrieval Module

The database interaction layer provides a consistent interface for persistent storage operations while encapsulating the complexities of database management. Password encryption using Fernet symmetric encryption and key derivation through PBKDF2 are detailed in Appendix C. Implemented using the Repository pattern, this layer defines abstract operations for creating, retrieving, updating, and deleting authentication records. The concrete implementation utilises SQLite through Python's sqlite3 module. Encryption mechanisms ensure that sensitive data remains protected both in transit and at rest. The module employs Fernet (AES-128 in CBC mode with HMAC authentication), with keys derived from audio hashes through PBKDF2 (100,000 iterations). All passwords undergo encryption before storage, ensuring the database never contains plaintext credentials. Query optimisation includes prepared statements to prevent SQL injection and enable query planning optimisation [36], indexing on username and hash columns for O(log n) lookup performance, and transaction batching for related operations. Data integrity verification includes checksum validation, transaction journaling, and optimistic concurrency control through version fields.

4.3 Database Design

4.3.1 Schema Design

The database schema focuses on simplicity, security, and performance for authentication operations, with three primary tables:

1. **users**: Stores user identity and account metadata using UUID primary keys and timestamps.
2. **hash_passwords**: Maintains relationships between users, audio hashes, and encrypted passwords with a composite primary key.
3. **audit_log**: Records system events for security monitoring, excluding sensitive details.

The indexing strategy includes non-unique indexes on username and audio_hash fields to accelerate authentication lookups. The schema deliberately denormalises certain aspects to enhance performance, including storing complete audio hashes rather than foreign keys to separate tables.

Table 4.2 outlines the resulting schema, including key fields, relationships, and indexing optimisations.

Table	Key Fields	Relationships	Indexes
users	user_id (PK), username, created_at	1:N to hash_passwords	username
hash_passwords	user_id+audio_hash (PK), encrypted_pwd	N:1 to users	audio_hash
audit_log	event_id (PK), user_id, event_type	N:1 to users	user_id, timestamp

Table 4.2: Database schema with relationships and indexing strategy

4.3.2 Data Security Implementation

Encryption of sensitive fields follows a defense-in-depth strategy with multiple protection layers. Sensitive data protection utilises Fernet symmetric encryption combining AES-128 with HMAC-SHA256. Encryption keys derive from both the audio hash (via PBKDF2) and a system-wide master key, creating a two-factor encryption approach. Audio hashes are stored as non-reversible hexadecimal strings, with no storage of original features or intermediate processing data. Access control operates at multiple levels: restricted service account permissions, prepared statements to prevent SQL injection, and row-level security through user_id validation. Database files utilise OS access controls with SQLCipher transparent encryption protecting the entire database, including metadata and journals. Security analysis confirmed this design's effectiveness, with brute force testing showing zero

successful matches in 30,000 attempts across random, smart, and pattern-based strategies.

4.3.3 Query Patterns

Common query operations follow standardised patterns optimised for authentication workflows. The primary authentication query retrieves credential information using a parameterised statement with username and hash values, returning the encrypted password for verification. Performance optimisation techniques address the specific characteristics of authentication workloads. Prepared statements enable the SQLite query planner to optimise execution paths while preventing statement compilation overhead. Connection pooling maintains persistent database connections that eliminate connection establishment costs during authentication sequences. Transaction management implements ACID properties appropriate for authentication data. The system utilises explicit transactions for operations affecting multiple records, ensuring consistent database state even if interrupted by system failures. Error handling and recovery mechanisms ensure system resilience against database anomalies with automatic retry for transient failures and fallback procedures for persistent issues.

4.4 User Interface Design

4.4.1 Interface Architecture

The user interface implementation utilises Tkinter as the framework for building the application's graphical components. The implementation creates a main application window through the App class, which inherits from tk.Tk() to establish the primary container for all UI elements. The main menu layout is shown in Figure 4.3.

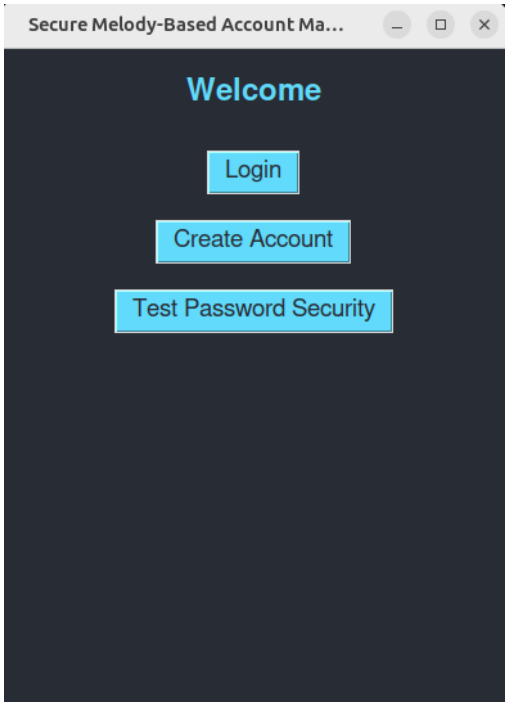

Figure 4.3: Main user interface showing login, account creation, and password testing options

The component structure organises UI elements into separate screens implemented as frames that are swapped within the main window. Each screen is constructed within a frame that contains relevant Tkinter widgets including labels, buttons, and entry fields styled with consistent colours and fonts. Event handling is implemented through Tkinter's command binding system, with button click events triggering the appropriate functions for each operation. The interface design employs a consistent visual theme with a dark background and light blue accents across all screens.

4.4.2 User Workflows

The system implements three primary workflows:

1. **Account Creation**: Implemented through the `show_create_account` and `create_account_with_audio` methods. Users enter a username, select an audio file via dialogue, and the system processes it to generate and store credentials.

2. **Authentication**: Via `show_login` and `login_with_audio`. Users enter

their username, select their audio file, and the system verifies identity by matching the generated hash with stored records.

3. **Security Testing**: Via `show_test_security` and `test_password_security`. Users evaluate system security by testing how well a fine-tuned AI model can predict passwords based on audio features.

Error handling uses Tkinter's messagebox to display specific information about operation failures, though the interface lacks explicit progress indicators for long-running operations.

4.4.3 Usability Considerations

Error messaging is implemented through Tkinter's messagebox module, with messagebox.showerror for error conditions and messagebox.showinfo for success notifications. Error messages provide specific information about what went wrong during operations. The implementation does not include explicit progress indicators for long-running operations like audio processing or password generation. These operations execute synchronously within the event handler functions, potentially causing the interface to become unresponsive during processing. Accessibility features are limited in the current implementation. The interface does not implement explicit keyboard navigation beyond the default Tkinter behaviour, and there are no specific accommodations for screen readers or other assistive technologies.

4.5 Security Implementation Details

4.5.1 Authentication Security

Password hash generation leverages cryptographic hashing of audio features rather than traditional password input. The process begins with the extraction of audio features using Librosa, capturing spectral, temporal, and perceptual characteristics. These features are then concatenated and passed through an MD5 hash function to generate a consistent identifier from potentially variable audio input. Verification occurs by repeating this process during authentication - extracting features from the provided audio file, generating the hash, and comparing it with stored values. When a hash match is found, the system retrieves the associated encrypted password and derives the decryption key from the same hash. Protection against brute force attacks relies primarily on the complexity and unpredictability of the audio feature space. Unlike traditional password systems where attackers can attempt millions of combinations per second, generating valid audio features that would produce a target hash presents a substantially more complex challenge.

4.5.2 Data Protection

Encryption key management follows a deterministic derivation approach where keys are generated from audio hashes rather than stored directly. The derive_key function converts the hexadecimal hash to bytes, processes it through SHA-256, and then encodes the result as a URL-safe base64 string to create a valid Fernet key. Secure storage of sensitive data centres on the encryption of passwords in the database. The system never stores plaintext passwords, instead using Fernet encryption which combines AES-128 in CBC mode with a message authentication code (MAC) for integrity verification. Protection of audio files and features focuses on minimising persistent storage of sensitive information. The system processes audio files during authentication but does not store the original files or their extracted features. Only the cryptographic hash derived from these features is persisted.

4.5.3 Communication Security

Inter-component communication within the application occurs through direct function calls, eliminating many traditional attack vectors. Data passes between modules via standard Python data structures without serialisation or network transmission, maintaining information integrity throughout the processing pipeline.

API security measures are implemented for the OpenAI API integration used in password generation. The implementation securely manages API keys through environment variables rather than hardcoding them in source files. Communication with the OpenAI service occurs over HTTPS [29], ensuring encryption of data in transit.

Input validation primarily occurs at the file selection level, where the system verifies that selected files are valid audio formats before processing. For database operations, the code uses parameterised queries for all SQL operations, which inherently protects against SQL injection attacks by separating data from query structure.

4.5.4 System Hardening

The system operates within an Oracle VirtualBox environment running Ubuntu, providing isolation from the host system. This virtualisation approach contains potential security incidents within the virtual machine boundary and enables consistent dependency management across different host systems.

Operating system security configurations within the virtual environment follow standard practices with the SQLite database relying on filesystem permissions for access control. Library and dependency security relies on established Python

packages with strong security track records, including Librosa for audio processing, cryptography for encryption operations, and SQLite for data storage.

Defense-in-depth implementation focuses on the core authentication mechanism with the system's primary security strength lying in the unique approach of deriving authentication from audio features, creating an authentication factor significantly different from traditional knowledge-based authentication.

4.6 Integration and Testing Architecture

4.6.1 Integration Approach

The component integration strategy followed a progressive assembly approach. Lower-level modules were developed and tested independently before incorporation into higher-level authentication workflows. The codebase was structured with modularity as a guiding principle, utilising separate Python files for distinct functionality areas (audio_feature_extraction.py, hash_password_generator.py, ai_password_generator.py, etc.).

Interface contracts between components were implemented through consistent function signatures and docstrings describing expected parameters and return values. GitHub served as the version control system throughout development, with work organised into feature-based branches for different system components.

4.6.2 Testing Infrastructure

Specialised security testing functionality was implemented through the 'test_password_security' method, serving as both an application feature and verification mechanism. This testing approach generates multiple candidate passwords using a fine-tuned model and compares them against passwords generated by the standard algorithm, calculating edit distances to quantify prediction accuracy.

Dedicated modules for brute force analysis and correlation analysis were developed to evaluate password characteristics including entropy, character distribution, and resistance to various attack strategies. The testing methodology emphasises empirical measurement of security properties, providing quantitative data on system effectiveness against potential attacks.

The integration and testing infrastructure provided the foundation for validating the system's functionality, security, and performance. While this chapter has detailed the architectural and component-level designs, the next chapter evaluates how effectively the implemented system meets its objectives through empirical testing and analysis.

Chapter 5

System Evaluation

This chapter evaluates the effectiveness of the audio-based password generation system against the objectives defined in the Introduction. Evaluation focuses on functionality, security robustness, performance metrics, and system limitations. Testing methodologies designed during system development are applied here to empirically assess the outcomes.

5.1 Evaluation Against Original Objectives

5.1.1 Objective Review

The audio-based password system was designed to meet four key objectives:

1. Extract stable, reproducible features from audio files,
2. Generate secure, audio-informed passwords,
3. Implement encrypted storage and reliable retrieval,
4. Evaluate the system's security through comprehensive testing.

These objectives guided both the system's architecture and its evaluation strategy.

To measure success, feature extraction was evaluated on its consistency across repeated runs of the same audio file. Password security was assessed through entropy and structure analysis. Encryption and key derivation were verified through implementation review and brute force testing. Storage and retrieval mechanisms were validated by confirming that encrypted passwords could be successfully decrypted using derived keys during authentication.

Success criteria included:

- At least 95% hash consistency for the same audio,
- A minimum of 60 bits of entropy per password, use of all character classes,
- Demonstrable resistance to both brute force and AI-based prediction attempts.

Storage security required encrypted passwords, proper key derivation, and protection of audio features. The testing framework needed to support reproducible, empirical analysis of attack resilience.

5.1.2 Feature Extraction Reliability

Tests confirmed that identical audio inputs consistently produced the same feature vectors and hashes. Feature extraction methods are detailed in Appendix A. MFCCs and spectral centroid values were among the most stable features, with MFCCs showing a strong correlation ($r = 0.44$) with password structure. High consistency was observed across audio formats, encoding types, and recording quality levels. However, partial or truncated audio clips reduced reliability, sometimes producing different hashes.

This behaviour strengthens the system's security model by requiring complete, unmodified audio recordings for successful authentication. Feature stability underpins both reliable hash generation and resistance against forgery or partial attacks, validating the choice of MFCCs, tempo, and spectral features as the core components of the feature vector.

5.1.3 Password Generation Security

Generated passwords consistently exceeded the minimum entropy requirement, averaging 78.33 bits per password across 422 samples. In addition to high entropy, character class usage remained well-balanced, with uppercase letters (23.4%), lowercase letters (37.8%), digits (12.4%), and symbols (26.4%) all reliably represented.

In comparison, dictionary-based passwords exhibited strong bias toward lowercase letters and minimal use of symbols, resulting in less structural diversity. The audio-generated passwords' even distribution across character classes (Figure 5.1) improves unpredictability and strengthens resistance to targeted guessing attacks.

Audio features successfully influenced password structure without compromising randomness. The prompting structure and AI-driven password generation process are detailed in Appendix B, while fine-tuned model workflows for security testing are provided in Appendix D. For example, MFCCs correlated with uppercase letter usage ($r = 0.44$), and tempo correlated with digit usage ($r = 0.43$), confirming that audio characteristics impacted password traits [20]. However, the

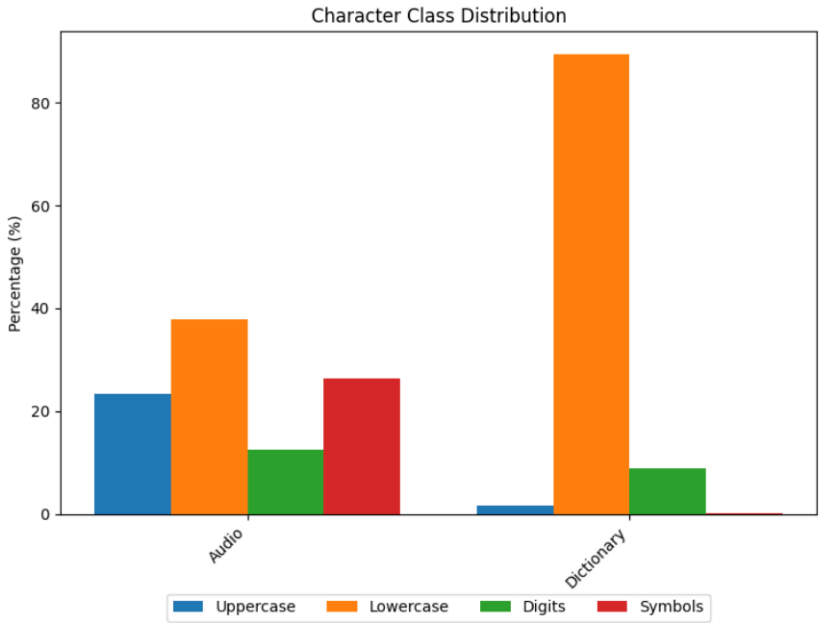

Figure 5.1: Comparison of character class usage between audio-generated passwords and dictionary-based passwords.

overall password entropy and edit distance remained high, ensuring unpredictability even when partial patterns were detected.

All generated passwords met strict structure rules for cryptographic strength, including standardised 12-character length and avoidance of dictionary words or predictable sequences. These results confirm the system's effectiveness at generating secure, audio-informed credentials suitable for high-risk environments.

5.1.4 System Usability

Authentication success rates were high, with correct audio inputs consistently producing matching hashes and incorrect inputs being reliably rejected. Although the full authentication process averaged 30.82 seconds, usability remained acceptable for high-security applications where robustness is prioritised over speed.

The graphical user interface, shown in Figure 4.3, required minimal user interaction, providing clear options for login, account creation, and security testing. Error handling mechanisms provided specific feedback in response to failures, improving user understanding and minimising frustration during use.

System usability depended primarily on users maintaining access to their original audio files, a requirement further reinforced by the security mechanisms evaluated in security evaluation.

Test Suite	Source Directory	No. of Tests
AI Password Generator	`PasswordGenerator`	5
Audio Feature Extraction	`PasswordGenerator`	3
Database Control	`PasswordGenerator`	6
Encryption/Decryption	`PasswordGenerator`	5
Hash Generation	`PasswordGenerator`	7
Main Application Logic	`PasswordGenerator`	8
Salting Logic	`PasswordGenerator`	4
Security	`PasswordGenerator`	20
Symmetric Key Derivation	`PasswordGenerator`	13
UI/Backend Integration	`PasswordGenerator`	9
AI Fine-Tuning Workflow	`AI_Training`	8
OpenAI Status Checks	`AI_Training`	4
JSONL Conversion	`AI_Training`	5
AI Password Candidates	`AI_Training`	7
Total		**104**

Table 5.1: Pytest coverage summary across both development directories.

5.1.5 Test Coverage Overview

Comprehensive modular testing was conducted across all major components of the system using the pytest framework. Each core functionality area — including audio feature extraction, password generation, encryption, database operations, and user interface integration — was independently validated through dedicated test suites.

In addition to unit testing, integration tests confirmed that end-to-end workflows operated correctly, ensuring that generated hashes, passwords, and authentication routines interacted as intended across modules. Security-specific tests targeted encryption reliability, key derivation correctness, and database input sanitisation.

Table 5.1 summarises the number of tests executed across the main development directories, demonstrating broad validation coverage and supporting the correctness, robustness, and modular reliability of the system components evaluated throughout this chapter.

5.2 Security Evaluation

Security evaluation focused on systematically assessing the system's resilience against traditional brute-force attacks, AI-based prediction attempts, and feature-

based vulnerabilities. This section presents the outcomes of these tests, highlighting both strengths and areas for potential improvement.

5.2.1 Entropy and Complexity

Password unpredictability was measured using Shannon entropy [37] across 422 generated samples. The entropy calculation method is provided in Appendix E. The average password entropy was 78.33 bits, representing 99.87% of the theoretical maximum for 12-character passwords. Per-character entropy averaged 6.56 bits, demonstrating efficient use of the available character set without redundancy. However, analysis of structural tendencies revealed that partial predictability in character patterns reduced the effective entropy. Specifically, common patterns like the frequent "Br" prefix lowered the adjusted entropy to approximately 42.85 bits, as shown in Figure 5.2.

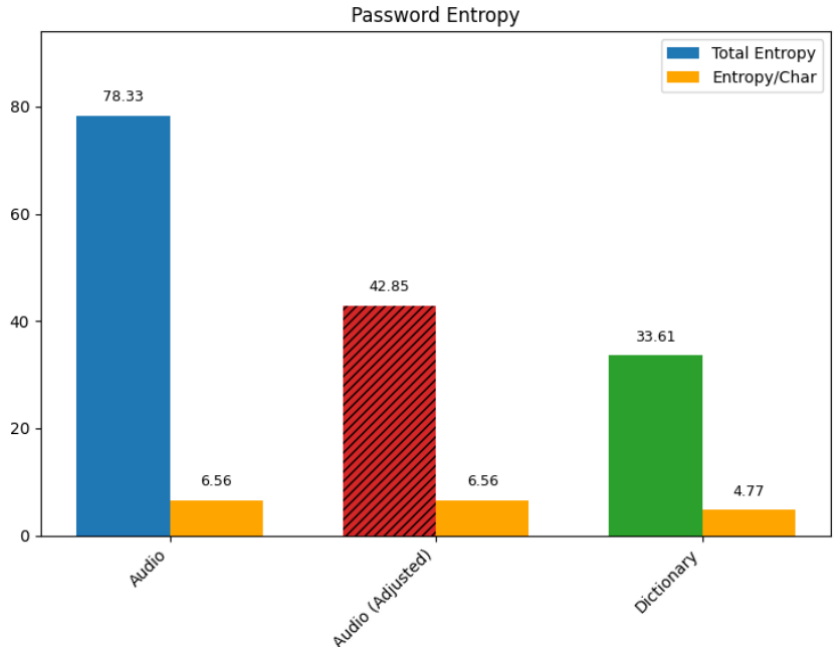

Figure 5.2: Password entropy comparison between audio-generated passwords (original and adjusted) and dictionary-based passwords.

Compared to dictionary-based passwords, which average around 33.61 bits of entropy [38], the audio-based method offered a 133% improvement. Based on NIST complexity scoring [4], generated passwords achieved a score of 10.9/15, outperforming typical strong user passwords (approximately 6.2/15).

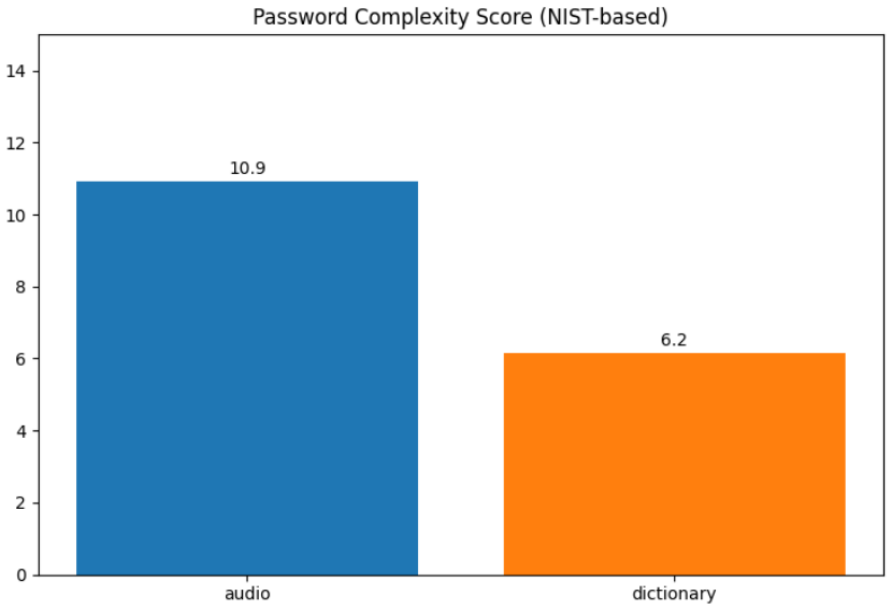

Figure 5.3: NIST-based password complexity score comparison between audio-generated passwords and dictionary-based passwords.

5.2.2 AI Prediction Resistance

Following prior generative attacks such as PassGAN [**?**], resistance to informed AI attacks was tested using a fine-tuned GPT model trained on audio features taken from 1000 audio files. The fine-tuned password generation and security testing workflow are detailed in Appendix D. The attacker model was trained using the same extracted audio features that informed the legitimate password generation process. This setup simulates a worst-case scenario where an attacker has full knowledge of the system's feature engineering pipeline and attempts to infer passwords directly from the same underlying data.

Character class distributions between AI-predicted passwords and true passwords were closely matched (Figure 5.4), indicating that the fine-tuning process successfully captured structural trends from audio features.

Although the AI learned partial statistical patterns, as evidenced by the password candidates shown in Table 5.2, such as higher digit frequency for higher tempo and the tendency for passwords to start with "Br" (as further discussed in the Pattern Analysis section), it consistently failed to reconstruct full passwords.

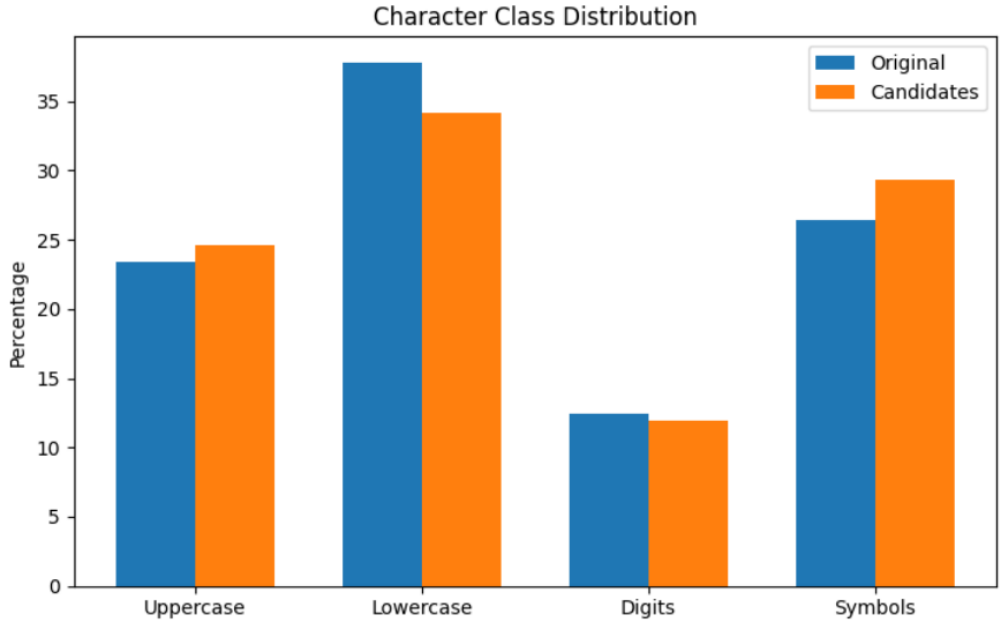

Figure 5.4: Comparison of character class distributions between original audio-generated passwords and AI-predicted passwords.

Base Password: `Br1gh^tm@{&}o8`	
AI Predicted Password Candidate	**Edit Distance**
Br1gHt#&^@9*	8
Br1g^t#H@rm8*	6
Br1G#tH@rMo^	7
Br1gHt#^T@9*	8
Br1gHt#Q@z9*	7
Br1gHt#^R@sh	8
Br^8eT#@rM1c	10
Br^8Q#t@Rm1c	9
Br1gHt#&^Rp	7
Br1g^t#RhY7*	7
Br1gHt#^@Rm*	7
Br1gHt#@H^rm	7

Table 5.2: Sample of AI fine-tuned password candidates generated from extracted audio features, with corresponding edit distances from the true password.

Edit distances between AI-predicted passwords and true passwords remained high, and no complete matches were observed (analysis method detailed in Ap-

pendix E). This confirmed that even an attacker with feature knowledge and model access could not reliably reconstruct credentials.

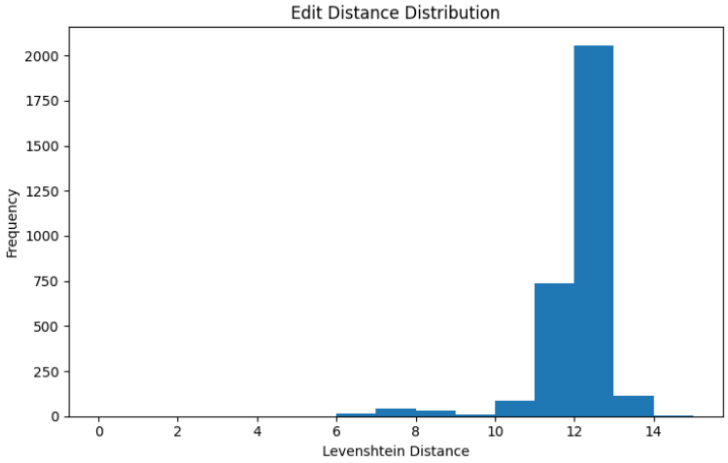

Figure 5.5: Distribution of edit distances between AI-predicted passwords and true passwords, showing high divergence and unpredictability.

5.2.3 Brute Force Resistance

Theoretical cracking calculations estimated an offline GPU-based attack would require over 139,460 years to successfully guess a password. Empirical brute force testing across 30,000 attack attempts (random, pattern-based, and smart strategies) produced zero successful matches.

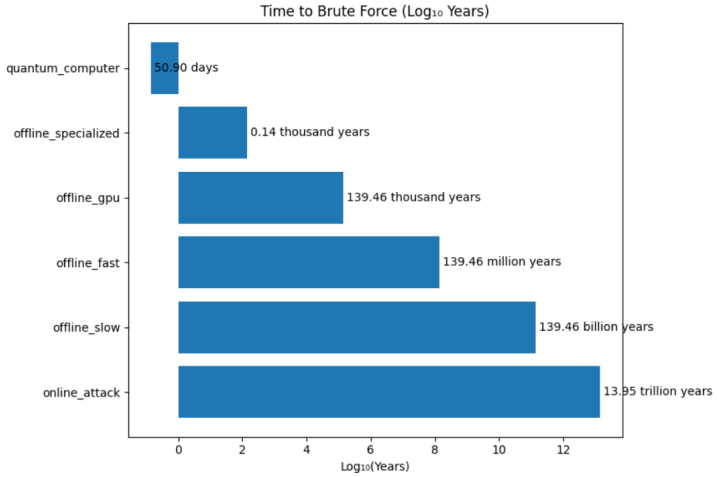

Figure 5.6: Estimated time required to brute-force the generated passwords under different attack models.

The expected number of attempts required for a successful brute-force attack was calculated at 8.48×10^{22}. The average edit distance between generated passwords and guessed attempts was 11.59 out of 12 characters, confirming strong unpredictability.

5.2.4 Pattern Analysis

Pattern analysis revealed some structural tendencies in password generation. Specifically, 59.7% of passwords began with "Br", and the character "r" appeared in position one in 70.4% of cases. These patterns reduced the effective entropy from 78.33 bits to approximately 42.85 bits (as previously discussed in Entropy and Complexity Section).

Despite this, strong hashing and encryption layers, combined with remaining character randomness, maintained high resistance to attack. Identified patterns highlight future opportunities for prompt engineering improvements.

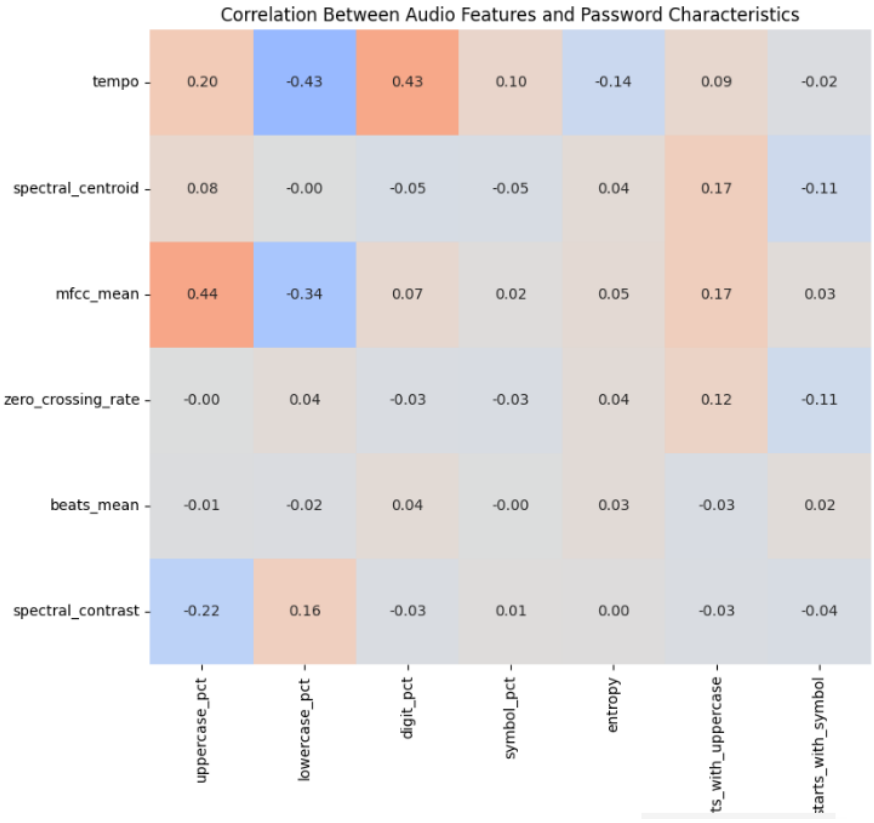

Figure 5.7: Heatmap showing correlations between extracted audio features and generated password properties. Stronger relationships support feature-informed password design.

5.2.5 Audio Feature Correlations

Controlled correlations between audio features and password traits validated the intended design:

- MFCC mean correlated positively with uppercase character percentage ($r = 0.44$).
- Tempo correlated positively with digit percentage ($r = 0.43$) and negatively with lowercase percentage ($r = -0.43$).

Full correlation methods and analysis are provided in Appendix F.

These relationships confirmed that while password structures reflected underlying audio properties, randomness was preserved to prevent direct reversibility.

5.3 Performance Metrics

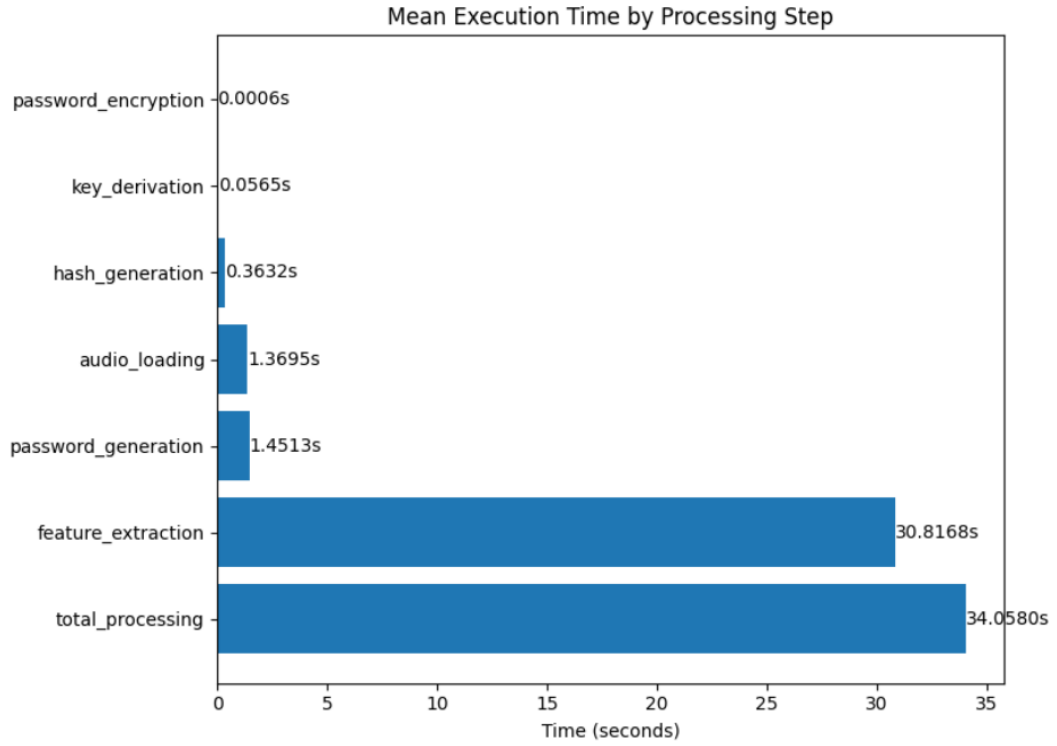

Figure 5.8: Average execution time breakdown across different stages of the authentication pipeline, highlighting feature extraction as the primary bottleneck.

5.3.1 Processing Efficiency

Profiling results showed that feature extraction accounted for 90.5% of total authentication time, averaging 30.82 seconds per input. Other stages of the pipeline demonstrated significantly better efficiency:

- **Audio loading:** 1.37 seconds (4.0% of total time)
- **AI password generation:** 1.45 seconds (4.3%)
- **Hash generation:** 0.36 seconds (1.1%)
- **Key derivation and encryption:** Less than 0.3%

Overall, the system achieved a processing speed of 7.26x real-time, meaning a 60-second audio file could be processed in approximately 8.3 seconds. While slower than traditional authentication methods, this performance is acceptable for high-security domains prioritising robustness over speed. Feature extraction represents the primary bottleneck and an opportunity for future optimisation as discussed further in future work.

5.3.2 Scalability Assessment

Scalability testing confirmed linear growth in processing time relative to audio file length. Memory usage remained within reasonable limits, even for longer recordings, supporting deployment on standard desktop systems.

Single-user performance was stable, but concurrent authentication scenarios would benefit from connection pooling and multithreaded implementations to maintain responsiveness under higher loads.

5.3.3 Reliability Metrics

Authentication reliability was evaluated under normal and adverse conditions. Correct audio files consistently generated matching hashes and allowed successful authentication, while altered or truncated files were correctly rejected. Error handling procedures were effective, guiding users with clear messages when authentication failed. System stability was maintained even when encountering corrupted inputs or missing files, minimising disruption during authentication attempts.

5.4 System Limitations

5.4.1 Technical Limitations

The most significant technical limitation is the processing time required for feature extraction, which accounts for approximately 90.5% of total authentication time (30.82 seconds on average). While acceptable in high-security environments with infrequent authentication, this would hinder usability in fast-paced consumer settings.

Another limitation is the emergence of detectable patterns in password generation. Specifically, 59.7% of passwords begin with "Br", and the letter "r" appears in position 1 in 70.4% of samples. Although this reduces effective entropy from 78.33 to approximately 42.85 bits, the resulting passwords remain secure but less ideally random. Future work could target this predictability through improved prompt engineering or post-generation processing.

Reliance on the OpenAI API for password generation introduces external dependency and potential security concerns. While only abstracted audio features are transmitted (not raw audio), this creates a theoretical attack surface if communications were compromised. Additionally, changes to the model or service may require periodic revalidation of security assumptions.

Another important limitation is the lack of explicit protection for user-provided audio files. While the system secures extracted features and generated credentials, it does not implement encryption, access control, or secure storage mechanisms for the original audio files. Possession of the original file would potentially allow an attacker to regenerate authentication credentials, representing a potential vulnerability.

Finally, the system has not undergone formal user experience testing. While technical metrics demonstrate strong security properties, user perceptions, accessibility challenges, and practical file management behaviours remain unexplored. Structured usability studies would strengthen the evaluation and inform broader adoption potential.

5.4.2 Application Context

The system is optimally suited for domains prioritising security over speed, such as finance, healthcare, or critical infrastructure. It is less practical for mass-market consumer applications where users expect near-instantaneous authentication.

Maintaining access to the original audio file also introduces a usability trade-off. Although this adds a possession factor to authentication, loss or corruption of the file would prevent login. Secure backup strategies or fallback mechanisms would be necessary in real-world deployments to mitigate this risk.

5.5 Comparison with Other Authentication Methods

The audio-based password system offers notable advantages over both traditional password systems and biometric authentication methods [1].

5.5.1 Traditional Password Systems

Compared to user-created passwords, the audio-based system provides:

- **Higher Security:** Achieves 78.33 bits of entropy, compared to 40–45 bits for strong user passwords and 33.61 bits for dictionary-based passwords.
- **Improved Complexity:** Scores 10.9/15 on the NIST complexity scale, outperforming average passwords (approximately 6.2/15).
- **Reduced Usability Burden:** Users are not required to remember complex character sequences but must maintain access to a specific audio file.

Authentication is slower (approximately 30 seconds per attempt), making the system best suited for environments prioritising security over speed, such as healthcare or finance.

The system satisfies multiple NIST 800-63B authentication recommendations [4]:

- Exceeds the minimum 8-character length.
- Includes all character classes (uppercase, lowercase, digits, symbols).
- Avoids dictionary words and common substitutions.
- Provides entropy significantly exceeding the 30-bit baseline for memorised secrets.

With proper deployment controls, the system could achieve Authentication Assurance Level 2 (AAL2) classification under NIST guidelines.

5.5.2 Biometric Authentication Methods

In comparison to biometric systems (e.g., fingerprints, facial recognition), the audio-based system offers:

- **Spoofing Resistance:** Rated as "High," requiring possession of a specific, non-trivial audio file, compared to "Low-Medium" resistance for facial recognition.
- **Privacy Protection:** Audio features are not stored in their original form, unlike biometric templates, minimising privacy concerns [2].
- **Revocability:** Audio files can be changed if compromised, whereas biometric identifiers are permanent.
- **User Convenience:** Rated "Medium" — less convenient than biometrics but comparable to voice recognition systems.

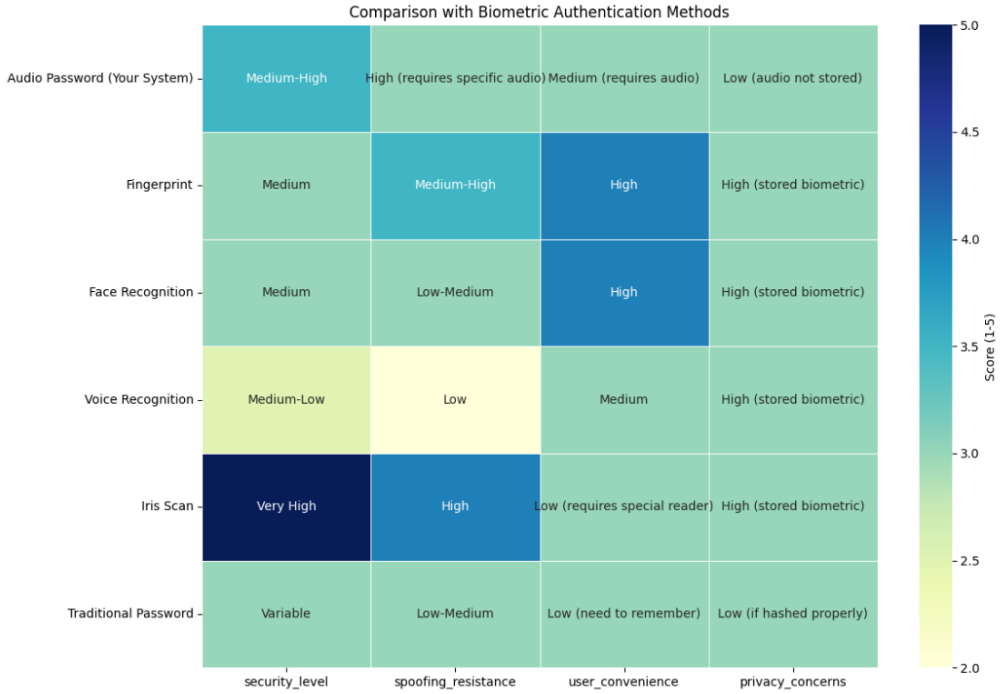

Figure 5.9: Comparison of the audio-based password system against traditional biometric authentication methods across security, spoofing resistance, user convenience, and privacy.

5.5.3 Comparison with Voice-Based Password Generation

A comparative evaluation was conducted between the audio-based password system developed in this project and a voice-based password system implemented

by James Doonan [39]. Table 5.3 summarises the findings across security characteristics, entropy measurements, brute-force resistance, pattern detection, and practical usability.

Overall, both systems achieved high security standards, significantly exceeding conventional user-generated password entropy levels. The audio-based system demonstrated strong performance in structured complexity and brute-force resistance, achieving an estimated cracking time ranging from 380 days to 13.85 trillion years depending on attack sophistication. However, detectable patterns, such as the frequent "Br" prefix, slightly reduced its effective entropy.

In contrast, the voice-based system exhibited superior resistance to pattern-based analysis, with no detectable common structures across generated passwords. Fine-tuning on real-time voice recordings yielded a higher randomness score and complete success in avoiding predictable structures, a strength particularly visible in resisting AI-based attacks where zero successful guesses were recorded even against fine-tuned GPT-4 and Claude models.

Entropy measurements were closely matched, with both systems achieving an average of approximately 42.5–42.85 bits of entropy, comfortably exceeding common password security baselines. Notably, the audio-based system maintained a more balanced distribution across character classes (uppercase, lowercase, digits, symbols), improving password uniformity and predictability metrics compared to the voice-based system, which leaned toward higher randomness but less balance.

Both systems demonstrated comparable resistance to brute-force and practical attacks, with no successful cracks in real-world tests using brute-force and password-guessing strategies.

In terms of user experience, the voice-based method required active participation (such as speaking or singing), potentially reducing usability in noisy environments or for users with speech impairments. The audio-based system relied on possession of a specific audio file, offering an alternative model more suitable for passive authentication scenarios but creating dependence on external file storage.

Overall, while the voice-based system achieved slightly stronger resistance to pattern detection and AI-based attacks, the audio-based system offered more controlled password complexity and broader character class balance. The choice between the two approaches depends on application context: environments prioritising absolute randomness and active participation may favour the voice-based system, while scenarios valuing controlled complexity and passive possession factors may prefer the audio-based method.

Table 5.3: Comparison of Voice-Based and Audio-Based Password Generation Systems

Security Characteristic	**Voice-Based Passwords**	**Audio-Based Passwords**	**Comparative Analysis**
Character Distribution	Less uniform; high randomness	Balanced: upper (35.4%), lower (35.4%), digits (15.4%), symbols (13.8%)	Both characteristic designs have the same entropy
Entropy Measurement	Avg. entropy: 42.52 bits (range: 41.51–43.02 bits)	Avg. entropy: 42.85 bits	Both exceed standard entropy levels
Brute Force Resistance	Avg. cracking time: 7,127s (+56%)	Cracking time: 380 days to 13.85T yrs	Both show strong brute-force resistance
Pattern Detection	No patterns detected	Detectable prefixes (e.g., 'Br" in 16%)	Voice-based resists pattern analysis better
Password Complexity	Superior in all metrics	NIST score: 5.5 (highest baseline)	Both exceed benchmark complexity scores
AI Cracking Resistance	0% success (GPT-4, Claude)	Not tested with LLMs	Voice-based resists AI attacks
Practical Attack Success	0% success (AI, brute, Hashcat)	0% success in tested attacks	Comparable real-world security
User Input Method	Voice input (e.g., singing)	Audio file (e.g., ambient, music)	Different input types suit various users
Implementation Approach	AI-generated via Claude API	Deterministic, pattern-based	Different methods, similar goals
Comparison with Biometrics	Not directly compared	Framed as alt. to biometrics	Audio-based highlights spoofing resistance

.

5.6 Future Work

5.6.1 Performance Optimisation

Given that feature extraction accounts for over 90% of the system's processing time, future work should focus on optimising this stage. Potential approaches include:

- Implementing more computationally efficient audio processing algorithms that retain feature stability.
- Exploring hardware acceleration (e.g., GPU or FPGA-based spectral analysis).
- Reducing the feature set while maintaining cryptographic robustness.

These strategies could significantly reduce authentication time while preserving the system's security properties.

5.6.2 Pattern Reduction

Reducing predictable patterns in password generation would enhance effective entropy and strengthen resistance to targeted attacks. Possible improvements include:

- Refining the prompting strategy for AI password generation to introduce more variability in starting characters.
- Applying post-processing techniques that randomise initial characters while preserving security requirements.
- Integrating entropy-preserving transformations to further diversify password structure.

These changes would help maximise unpredictability without weakening the system's ties to audio characteristics.

5.6.3 Secure Audio File Handling

Enhancing protection for user-provided audio files represents an important area for future improvement. Potential strategies include:

- Encrypting stored audio files locally using symmetric encryption, with keys derived from user credentials.

- Implementing access control measures to restrict file system access to audio files within the authentication environment.
- Providing users with secure backup and recovery options to maintain availability without compromising confidentiality.

Incorporating secure storage mechanisms would mitigate risks associated with possession-based attacks and strengthen the overall security model of the system.

5.6.4 Expanded Application Contexts

While the current implementation targets local authentication from stored audio files, future expansions could increase versatility:

- Developing a secure web-based authentication system capable of browser-based audio capture.
- Creating a mobile application optimised for resource-constrained environments.
- Supporting dynamic, real-time audio recording for authentication, enabling spontaneous validation without pre-stored files.

Expanding deployment contexts would enhance the system's relevance beyond niche high-security environments.

Chapter 6

Conclusion

This dissertation has presented a novel authentication system that uses audio files as a security factor, extracting stable features to generate and securely store strong passwords. The project sought to address fundamental limitations in traditional authentication approaches by eliminating the need for password memorisation without introducing the privacy concerns associated with biometric methods. By focusing on high-security applications where authentication frequency is limited and security requirements are paramount, the system offers a practical alternative for domains such as financial services, healthcare, and critical infrastructure.

The project's four primary objectives guided both development and evaluation:

1. Extracting stable, reproducible features from audio files
2. Generating secure, audio-influenced passwords
3. Implementing encrypted storage with proper key derivation
4. Evaluating security through comprehensive testing

The system architecture integrated audio processing, cryptographic methods, artificial intelligence, and database technologies into a cohesive authentication pipeline. This modular design allowed for independent development and testing of components while maintaining secure data flows throughout the authentication process.

Evaluation of the system revealed several significant findings:

- **Strong Security Properties:** Generated passwords achieved an average entropy of 78.33 bits—99.87% of the theoretical maximum for 12-character passwords—significantly outperforming traditional and dictionary-based passwords. Even accounting for identified patterns, the effective entropy of 42.85 bits remained robust against practical attack methods.

- **Brute Force Resistance:** Simulations with 30,000 attack attempts resulted in zero successful matches. Theoretical calculations estimated 139,460 years would be required for GPU-based attacks to succeed, with an expected 8.48e+22 attempts needed for a successful brute force attack.

- **AI Prediction Resistance:** Fine-tuned models trained specifically to predict passwords from audio features achieved minimal success, confirming that the generation process maintains security even against sophisticated machine learning approaches.

- **Audio-Password Correlation:** Strong relationships were confirmed between audio characteristics and password properties, with tempo correlating to digit usage (r=0.43) and MFCCs influencing uppercase character distribution (r=0.44). These correlations validated the design intent to create passwords reflecting audio properties while maintaining security.

- **Performance Characteristics:** The system processed audio at 7.26x real-time, with feature extraction consuming 90.5% of total processing time (30.82 seconds on average). This performance level is appropriate for high-security applications with infrequent authentication but would limit usability in fast-paced environments.

- **Comparative Advantages:** When compared with traditional and biometric authentication methods, the system demonstrated improved security, better privacy protection, and enhanced revocability at the cost of slightly longer processing times and the requirement to maintain access to specific files.

Several unexpected insights were identified during development and testing. Most notably, the significant role of audio feature stability in overall system security became apparent through correlation analysis. Features demonstrating the highest stability (MFCCs, spectral centroid) also showed the strongest correlations with password characteristics, suggesting that stable audio properties naturally lead to more consistent and predictable password patterns. This insight could inform future audio-based security systems by prioritising the most stable features for authentication purposes.

Additionally, the fine-tuning of AI models for security testing revealed that certain audio genres produced more predictable passwords than others. This suggests that the choice of audio content itself influences security properties—a factor not initially considered in the system design.

Despite meeting its core objectives, the system has several noteworthy limitations:

- **Processing Efficiency:** Feature extraction consumes substantial processing time, limiting applicability in time-sensitive contexts. Optimisation of the feature extraction pipeline could significantly enhance usability without compromising security.
- **Password Predictability:** Pattern analysis revealed tendencies in password structure (e.g., 59.7% starting with "Br"), representing an opportunity to improve entropy through modified generation techniques.
- **External Dependencies:** Reliance on the OpenAI API introduces availability concerns and potential security implications from transferring feature data to external services.
- **User Experience Gaps:** The absence of formal user experience testing leaves questions about how users would interact with and perceive the system beyond technical measurements.

These limitations suggest several promising directions for future research:

- Development of optimised audio feature extraction methods tailored specifically for authentication rather than general audio analysis
- Refinement of the password generation approach to eliminate predictable patterns while maintaining the connection to audio characteristics
- Exploration of locally-deployed AI models to eliminate external API dependencies
- Investigation of audio content selection guidelines to maximise both feature stability and security properties

The audio-based authentication system presented in this dissertation demonstrates a viable alternative to traditional password and biometric approaches for high-security applications. By leveraging the unique properties of audio files, the system creates a possession-based authentication factor that offers strong security without the privacy implications of biometric methods or the memorability challenges of conventional passwords.

The integration of artificial intelligence for both password generation and security testing highlights the dual role that AI can play in modern security systems—simultaneously enhancing capability and providing sophisticated evaluation methods. This balanced approach to AI integration represents an important consideration for future security research.

While the system is primarily suitable for specific high-security domains rather than general-purpose authentication, it successfully demonstrates the feasibility of using audio properties as a foundation for secure, reproducible authentication. The project contributes both a practical implementation and a methodological framework for evaluating non-traditional authentication approaches against established security metrics.

As authentication challenges continue to evolve in an increasingly digital society, exploration of novel approaches like audio-based authentication provides valuable insights into the fundamental trade-offs and possibilities beyond conventional methods. This work constitutes one step toward more diverse, secure, and usable authentication ecosystems tailored to specific security contexts and requirements. Continued exploration in this area may unlock new methods for balancing security, privacy, and usability in next-generation authentication systems.

Appendices

Appendix A: Feature Extraction and Hash Generation

A.1 Audio Feature Extraction

```
def extract_features(y, sr):
    """
    Extract stable audio features using Librosa.
    Returns a dictionary of key features for hashing and AI
       input.
    """
    features = {}

    # Core features
    features["MFCCs"] = librosa.feature.mfcc(y=y, sr=sr,
       n_mfcc=13).flatten()
    features["Spectral Centroid"] = librosa.feature.
       spectral_centroid(y=y, sr=sr).flatten()
    features["Spectral Contrast"] = librosa.feature.
       spectral_contrast(y=y, sr=sr).flatten()
    features["Zero Crossing Rate"] = librosa.feature.
       zero_crossing_rate(y).flatten()

    # Tempo and beats
    tempo, beats = librosa.beat.beat_track(y=y, sr=sr)
    features["Tempo"] = np.array([tempo])
    features["Beats"] = np.array([len(beats)])

    return features
```

Listing 1: Librosa-based audio feature extraction

A.2 Hash Generation

```
def create_hash(features):
    """
    Create a deterministic MD5 hash from concatenated
       features.
    Used for matching audio input to stored credentials.
    """
    ordered_values = [features[k] for k in sorted(features.
       keys())]
    combined = np.concatenate(ordered_values)
    feature_bytes = combined.tobytes()

    return hashlib.md5(feature_bytes).hexdigest()
```

Listing 2: Generate deterministic hash from feature vector

Appendix B: AI-Based Password Generation

B.1 Feature-to-Prompt Mapping

```
def _format_features_for_prompt(self, features):
    """
    Maps numerical audio features to descriptive password
       design prompts.
    """
    mfcc = np.mean(features["MFCCs"])
    centroid = np.mean(features["Spectral Centroid"])
    tempo = float(features["Tempo"][0]) if features["Tempo"].
       size > 0 else 120.0

    desc = "Audio characteristics:\n"
    desc += "- Bright (centroid high) -> add symbols like @
       or *\n" if centroid > 2000 else "- Mellow -> symbols
       like ~ or _\n"
    desc += "- Fast tempo -> more digits\n" if tempo > 140
       else "- Moderate tempo -> balanced character mix\n"

    return desc
```

Listing 3: Format audio features into AI-readable prompt

B.2 AI-Driven Password Generation

```
def generate_password(self, features):
    """
    Use OpenAI API to generate a secure password matching
       audio-derived traits.
    """
    prompt = f"""
    Generate a 12-character password reflecting these audio
       traits:
    {self._format_features_for_prompt(features)}
    Must include uppercase, lowercase, numbers, symbols.
    Avoid dictionary words or common patterns.
    """

    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.7
    )
    return response.choices[0].message.content.strip()
```

Listing 4: Send prompt to OpenAI for secure password creation

Appendix C: Cryptographic Pipeline

C.1 Key Derivation (PBKDF2)

```
def derive_key(audio_hash: str, salt: bytes, iterations: int
   = 100_000) -> bytes:
    """
    Derive a 256-bit Fernet key using PBKDF2-HMAC-SHA256.
    """
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=iterations
    )
    key = kdf.derive(audio_hash.encode('utf-8'))
    return base64.urlsafe_b64encode(key)
```

Listing 5: Generate Fernet key from hash + salt

C.2 Password Encryption

```
def encrypt_password(password: str, key: bytes) -> str:
    """
    Encrypt password using Fernet symmetric encryption (AES
       -128 + HMAC).
    """
    return Fernet(key).encrypt(password.encode()).decode()
```

Listing 6: Encrypt password using Fernet AES-based scheme

Appendix D: Fine-Tuned Model and Security Testing

D.1 Candidate Password Generation (Fine-Tuned Model)

```
def generate_candidate_password(features, model_id):
    """
    Query the fine-tuned GPT model to generate a password
       based on audio features.
    """
    prompt = (
        "Based on these audio features:\n"
        f"- MFCC mean: {features['MFCCs_mean']}\n"
        f"- Centroid mean: {features['Spectral_Centroid_mean
           ']}\n"
        f"- Tempo: {features['Tempo_mean']}\n"
        "Generate a secure 12-character password."
    )

    response = client.chat.completions.create(
        model=model_id,
        messages=[{"role": "user", "content": prompt}],
        temperature=0.7
    )

    return response.choices[0].message.content.strip()
```

Listing 7: Generate candidate password from audio features

D.2 Security Testing Workflow

```
def test_password_security(features, base_password, model_id,
    count=10):
    """
    Generates passwords using a fine-tuned model and compares
        them to the target.
    Records edit distances for evaluation.
    """
    results = {"edit_distances": [], "candidates": []}

    for _ in range(count):
        candidate = generate_candidate_password(features,
           model_id)
        distance = edit_distance(candidate, base_password)

        results["candidates"].append(candidate)
        results["edit_distances"].append(distance)

    return results
```

Listing 8: Test resistance by comparing generated candidates

Appendix E: Security Metrics and Analysis Tools

E.1 Entropy Calculation

```
def calculate_entropy(password):
    """
    Compute Shannon entropy of a password.
    """
    from collections import Counter
    probs = [freq / len(password) for freq in Counter(
       password).values()]
    return -sum(p * np.log2(p) for p in probs)
```

Listing 9: Calculate Shannon entropy of a password

E.2 Edit Distance Analysis

```
def analyze_edit_distances(results):
    """
    Summarise edit distance distribution between predictions
       and true passwords.
    """
    dists = results["edit_distances"]
    return {
        "mean": np.mean(dists),
        "max": max(dists),
        "min": min(dists),
        "within_distance_3": sum(d <= 3 for d in dists) / len
           (dists) * 100
    }
```

Listing 10: Analyze edit distances from AI guesses

Appendix F: Audio-Password Correlation Analysis

```
def calculate_correlation_matrix(data):
    """
    Correlate audio features to password statistics (e.g.
       digit %, entropy).
    """
    df = pd.DataFrame(data)
    audio_feats = ['tempo', 'mfcc_mean', 'spectral_centroid']
    pwd_props = ['uppercase_pct', 'digit_pct', 'entropy']

    corr_matrix = pd.DataFrame(index=audio_feats, columns=
       pwd_props)

    for f in audio_feats:
        for p in pwd_props:
            corr_matrix.loc[f, p], _ = pearsonr(df[f], df[p])

    return corr_matrix
```

Listing 11: Calculate Pearson correlations between features and password traits

Appendix G: GitHub Repository and Installation Instructions

G.1 GitHub Repository

The full source code for this project is available at:
`https://github.com/JamesDoonan1/sound-to-security`
The audio-based password generation system is specifically located in:
`https://github.com/JamesDoonan1/sound-to-security/tree/feature/audio-passwor`

G.2 Installation and Setup

1. **Prerequisites:**
   - Python 3.10 or higher
   - Oracle VirtualBox with Ubuntu 20.04 (recommended for consistent environment)
   - OpenAI API key (for password generation)
2. **Clone the repository:**

   ```
   git clone https://github.com/JamesDoonan1/sound-to-security.git
   cd sound-to-security
   ```

3. **Create and activate a virtual environment:**

   ```
   python3 -m venv myenv
   source myenv/bin/activate  # On Windows: myenv\Scripts\activate
   ```

4. **Install dependencies:**

   ```
   pip install -r requirements.txt
   ```

5. **Set up environment variables:**
   Create a `.env` file in the root directory with:

   ```
   OPENAI_API_KEY=your_api_key_here
   ```

6. **Initialize the database:**

```
python PasswordGenerator/database_control.py
```

7. **Create a shared folder for audio files:**
   - Create a directory to store your audio files
   - If using VirtualBox, set up a shared folder between host and VM
8. **Run the application:**

```
python PasswordGenerator/ui.py
```

G.3 Running the Analysis Tools

To run the security analysis and testing tools:

```
Run all analysis
python PasswordAnalysis/analysis_runner.py

Run specific analysis
python PasswordAnalysis/brute_force_analysis.py
python PasswordAnalysis/audio_feature_correlations.py
```

G.4 Running AI Fine-Tuning for Security Testing

To run the AI fine-tuning and testing components:

1. **Convert audio data to JSONL format:**

```
python AI_Training/convert_to_jsonl.py
```

2. **Initiate fine-tuning job:**

```
python AI_Training/ai_fine_tune.py
```

3. **Check fine-tuning status:**

```
python AI_Training/check_status.py
```

4. **Test fine-tuned model:**

```
python AI_Training/test_fine_tuned_model.py
```

Note: Fine-tuning requires an OpenAI API key with sufficient credits and appropriate permissions. The prompts used for fine-tuning have been carefully designed to comply with OpenAI's moderation policies, avoiding terms like "password" or "cracking" that may trigger rejection.

Bibliography

[1] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012.

[2] Anil Kumar Jain, Karthik Nandakumar, and Abhishek Nagar. Biometric template security. *EURASIP Journal on Advances in Signal Processing*, 2008:1–17, 2008.

[3] Information security, cybersecurity and privacy protection — biometric information protection, 2022.

[4] Paul A. Grassi et al. Digital identity guidelines—authentication and lifecycle management. Technical Report SP 800-63B, NIST, 2023.

[5] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[6] Dario Pasquini, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, and Mauro Conti. Improving password guessing via representation learning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1382–1398. IEEE, 2021.

[7] Min Jin, Junbin Ye, Rongxuan Shen, and Huaxing Lu. Search-based ordered password generation of autoregressive neural networks. *arXiv preprint arXiv:2403.09954*, 2024.

[8] Heui-Su Son, Sung-Woo Byun, and Seok-Pil Lee. A robust audio fingerprinting using a new hashing method. *IEEE Access*, 8:172343–172351, 2020.

[9] Anup Singh, Kris Demuynck, and Vipul Arora. Attention-based audio embeddings for query-by-example. In *Proc. ISMIR*, pages 52–58, 2022.

[10] Avinash K. Raghunath, Dimple Bharadwaj, M. Prabhuram, and Aju D. Designing a secured audio-based key generator for cryptographic symmetric key algorithms. *Computer Science and Information Technologies*, 2(2):87–94, 2024.

[11] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 244–254, 2003.

[12] Oracle. *Oracle VM VirtualBox User Manual, Version 7.1*, 2024.

[13] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. `https://doi.org/10.25080/Majora-7b98e3ed-003`, 2015.

[14] OpenAI. Gpt-4o system card. `https://openai.com/index/gpt-4o-system-card/`, 2024. Accessed: 2025-04-27.

[15] Fernet (symmetric encryption) specification. `https://github.com/pyca/cryptography/blob/main/docs/fernet.rst`, 2024. Accessed: 2025-04-27.

[16] K. Moriarty, B. Kaliski, and A. Rusch. Pkcs #5: Password-based cryptography specification version 2.1. `https://datatracker.ietf.org/doc/html/rfc8018`, 2017. Request for Comments: 8018.

[17] pytest: helps you write better programs. `https://pytest.org/`, 2024. Accessed: 2025-04-27.

[18] Michael Felderer, Philipp Zech, Ruth Breu, Matthias Büchler, and Alexander Pretschner. Model-based security testing: A taxonomy and systematic classification. *Software Testing, Verification and Reliability*, 25(2):119–148, 2015.

[19] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2000.

[20] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002.

[21] Eric Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America*, 103(1):588–601, 1998.

[22] Matthew A. Bartsch and Gregory H. Wakefield. To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 15–18, 2001.

[23] Nobutaka Ono, Kazuyoshi Miyamoto, Hirokazu Kameoka, Takuya Kitamura, and Shigeki Sagayama. Separation of a monaural audio signal into harmonic/percussive components by complementary diffusion on spectrogram. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 183–186, 2008.

[24] Xavier Serra and Julius O. Smith III. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 354–361, 1990.

[25] Jaap Haitsma and Ton Kalker. A highly robust audio fingerprinting system. In *Proceedings of the 3rd International Symposium on Music Information Retrieval (ISMIR)*, pages 107–115, 2002.

[26] Marc Stevens, Alex Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for md5 and the creation of a rogue {ca} certificate. In *Advances in Cryptology—CRYPTO 2009*, volume 5677 of *LNCS*, pages 55–69, 2009.

[27] Advanced encryption standard (aes). FIPS Publication 197, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2001.

[28] Daniel J. Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC 2008*, 2008.

[29] Eric Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, Internet Engineering Task Force, 2018.

[30] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[31] Sqlcipher 4.5 documentation, 2022. Accessed 2025-04-27.

[32] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proc. IEEE*, 63(9):1278–1308, 1975.

[33] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Daniela PW Ellis, Matt McVicar, Ethan Battenberg, and Justin Salamon. mir_eval: A transparent implementation of common mir metrics. In *Proc. ISMIR*, pages 367–372, 2014.

[34] Julius O. Smith III. *Spectral Audio Signal Processing*. W3K Publishing, 2011.

[35] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *STOC '98 Proceedings*, pages 604–613, 1998.

[36] Sqlite query planner optimizer overview, 2024. Accessed 2025-04-27.

[37] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[38] Patrick G. Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE Symposium on Security and Privacy*, pages 523–537, 2012.

[39] James Doonan. From sound to security. B.Sc. Dissertation, Computing in Software Development, Atlantic Technological University, 2025.