CTEC3451 - P2665421, Luke Dawson

Supervisor - Mehmet Kiraz

Final Deliverable

Privacy-Preserving Biometric Authentication

Abstract

In current times, the security and privacy of personal data are key, particularly within biometric authentication systems. The traditional authentication methods are becoming increasingly more insufficient against sophisticated cyber threats, leading to the adoption of biometric systems. While these systems do offer enhanced security, they also raise significant privacy concerns due to the unchangeable nature of biometric data. This project introduces a Privacy-Preserving Biometric Authentication system which leverages a Fully Homomorphic Encryption scheme to further enhance the security and privacy of biometric data.

The system's cryptographic technique allows for computations to be performed on encrypted data without decryption. This method keeps biometric data encrypted throughout the authentication process, protecting it against unauthorised access and breaches. The system is designed with a robust back-end for cryptographic operations and a simplistic user-friendly front-end interface designed using Tkinter and Python.

Using an agile process methodology, this system helps enhance biometric authentication technologies with future implementations aiming to improve computational efficiency and advanced user interaction. This project has helped set new standards by integrating sophisticated encryption methods, preserving privacy, and ensuring integrity within biometric systems.

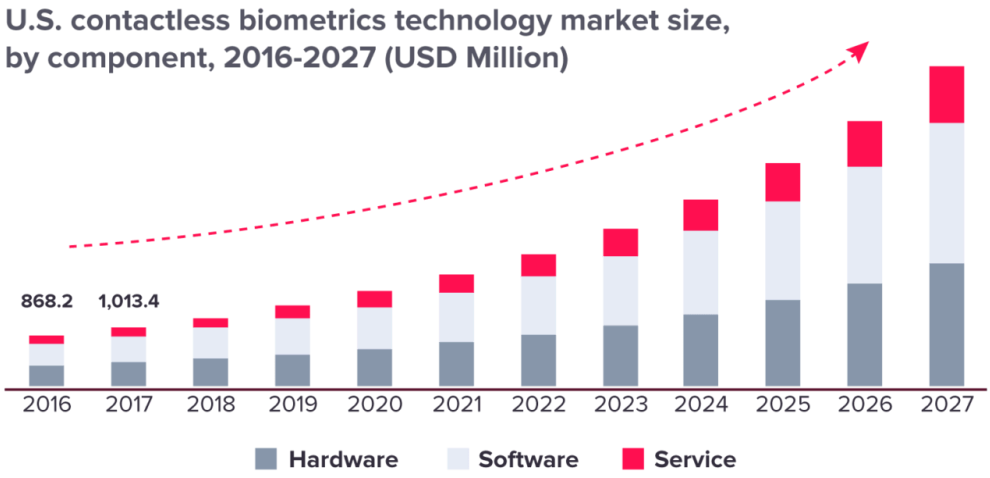
Acknowledgement

I extend my deepest gratitude to Dr. Mehmet Kiraz for his invaluable guidance and support throughout the duration of this project. His advice and insightful feedback were instrumental in shaping this project.

Introduction

In this modern new digital age, keeping the security and privacy of personal data is a key concern. With the introduction of sophisticated cyber threats, older, more traditional authentication methods have increasingly been deemed inadequate *(Jain, Ross & Nandakumar, 2016)*. Biometric authentication creates unique physiological or behavioural characteristics, however, considering the nature of biometric data - with its uniqueness - it creates significant privacy concerns. Once compromised, the data is unable to be revoked or reissued, making its protection critical for user safety (Acquisti, Friedman, & Telang, 2006).

Figure 1 - The History of Biometrics - Recfaces (See Figure 1, References / Figures)



The integration of biometric systems in various fields, from smartphones to banking, highlight the growth of its significance. Yet, while these systems become more noticeable, the risks, data breaches, and unauthorised surveillance that are aligned with these systems become more noticeable to *(Cavoukian, 2012)*. These issues have sparked a need for new technologies to ensure that the integrity and confidentiality of biometric data throughout the authentication process is kept. This brings the concept of Privacy-Preserving Biometric Authentication (PPBA), a newer approach that is designed to help safeguard users' biometric information from potential attacks. The core of PPBA is with its encryption method - Fully Homomorphic Encryption (FHE) - a groundbreaking cryptographic technique that allows for computations on encrypted data without requiring the data to be decrypted *(Gentry, 2009)*. FHE represents a paradigm shift, which creates a secure processing of sensitive information while keeping its encrypted state, preserving the users' privacy and improving the security.

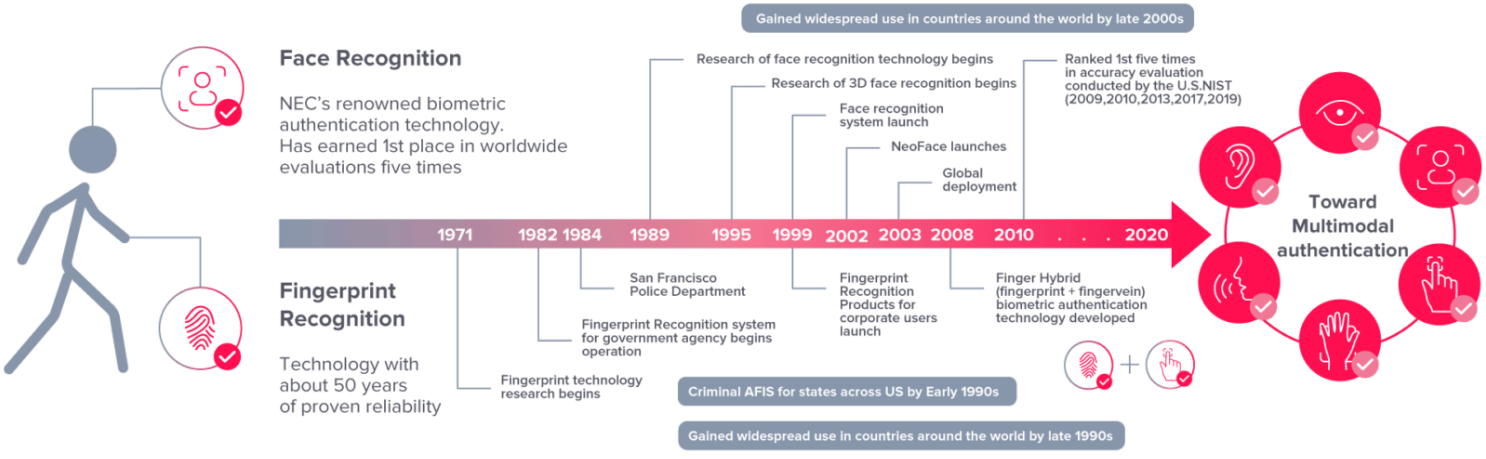
This project aims to design and implement a PPBA system leveraging the power of FHE. By doing so, I will address the challenges of securing biometric data and preserving the users privacy, creating a new standard for authentication technologies. the project has allowed me to explore the potential of FHE within the world of biometric authentication, highlighting its effectiveness, challenges, future applications and implementations (Naehrig, Lauter, & Vaikuntanathan, 2011). Integrating FHE, in a PPBA system will not only aim to protect against unauthorised access and breaches but also foster trust within biometric authentication as a secure and conscious method when verifying identity (Boneh, Gentry, Halevi, Wang, & Wu, 2013).

Background of Biometric Integration

Historical Overview of Biometric Authentication

In previous years biometric authentication has slowly transitioned from physical measurements to sophisticated and digital, revolutionising personal identification and security protocols. The roots of these processes can be traced all the way back to ancient civilisations, itis only recently that we are seeing a leap with the automation of fingerprint identification systems (Maltoni, Maio, Jain, & Prabhakar, 2009). Today biometrics encompass a wide variety of technologies, including facial recognition, iris scans, and voice identification, creating crucial roles in national security, healthcare, and personal device security.

Figure 2 - The History of Biometrics - Recfaces (See Figure 2, References / Figures)



The Privacy and Security Challenges in Biometric Systems

As biometric systems have become more prominent, the importance of privacy and security concerns have risen. The irreversibility of biometric data means a breach has further consequences for individuals privacy and security (Jain, Nandakumar, & Ross, 2016). Data breaches have become more high-profiled and begun to highlight the vulnerabilities that occur in biometric databases, highlighting the urgent need for protection mechanisms in order to help defend against both external attacks and inside threats (Acquisti, Taylor, & Wagman, 2016).

Introduction to Privacy-Preserving Techniques

In response to these challenges, the field has gravitated towards prioritising privacy-preserving techniques when making biometric systems. These techniques are used in tandem with systems to secure biometric data during its lifecycle, from collection to storage and analysis, ensuring that the user's privacy is maintained without losing integrity during the authentication process (Rane, Draper, & Boufounos, 2013).

Motivational Problems for the project

Privacy Risks

Biometric data is extremely personal and its exposure can lead to significant privacy violations. Unlike passwords, biometric traits are linked to personal individuals and are incapable of changing meaning a compromise creates a irreversible loss. The risks expand beyond unauthorised access though as they create potential issues for misuse of surveillance. Furthermore Data breaches in biometric databases also expose individuals to risks of identity theft that are difficult to fix, as you are unable to alter biometric data as you would a password. Due to these issues this system emphasises the encryption of biometric data, not only to prevent unauthorised access but to also keep privacy in the event of a system breach.

Security Vulnerabilities

My PPBA system is designed to help mitigate risks - such as attacks - by encrypting data at every step. This is done through using Fully Homomorphic Encryption - a version of homomorphic encryption - that allows biometric data to be remained encrypted even during processing, removing risks associated with interception of any data. However, vulnerabilities could also appear if the encryption keys are found or if there is a flaw in the implementation of TenSEAL.

Lack of User Control

A major concern in biometric systems is that users have a lack of control over their own data. In most normal systems once biometric information is stored users don't have a say over how it is used or who has access to it. By encrypting the biometric data, the PPBA system ensures that users data remains secure.

Product Overview

The PPBA system represents a transformative approach to biometric security through the integration of Fully Homomorphic Encryption (FHE). This system helped address the critical challenge of maintaining user privacy and security, as biometric authentication has now become a pillar of access control and identity verification.

Figure 3 - The History of Biometrics - Recfaces (See Figure 3, References / Figures)

The main point of using FHE allows for complex mathematical operations to be performed on the encrypted data then when compared, it matches the outcome of operations performed on the plaintext. By applying FHE to biometric authentication, the system ensures that the authentication process - comparing user biometric data with data stored - occurs without needing to decrypt the

biometric information. This protects the users data from potential attacks but also complies with data protection regulations.

The systems architecture is designed to be robust, capable of allowing for various biometric inputs while also being adaptable for future advancements in encryption and biometric equipment. The back-end will focus on the cryptographic operations essential for a system that uses privacy-preserving techniques and computations, while the planned front-end will provide users with a seamless gui throughout its operatoins.

Due to lack of equipment, I am unable to retrieve or process actual biometric samples or data. The system will instead simulate the capture, encryption and authentication of biometric data through a string of integers. This prototype will lay the groundwork for a comprehensive system that can be extended to real-world applications, creating a system that is secure and private biometric authentications can be processed across different sectors, including finance, healthcare, and access control systems.

Looking forward, the system will gauge new avenues for research and development opportunities in privacy-preserving technologies. By addressing the current challenges of different biometric authentication systems - focusing on user privacy and security - the system solution stands at the front of a new era in digital security.

Design + System Components

System Functionality

The core functionality of the Privacy-Preserving Biometric Authentication System relies on the ability to process and authenticate biometric data without exposing the raw data at any point. The system's primary operations include encrypting biometric data using the fully homomorphic encryption scheme, the secure storage of the data, and the privacy-preserving authentication process. This functionality allows computations needed for authentication to be performed on encrypted data, guaranteeing that user privacy is maintained.

User Types

The current iteration of the system caters to two user types - with the application for a future third user: Administrators, Base Users, and Guest Users. Administrators hold the highest tier of security, able to manage user accounts, system settings, and reviewing the logs of the system. Base Users are the core demographic of the system holding the majority of user types, capable of registering and authenticating their biometric data, which is then encrypted and stored for future authentication and use. For future implementations a Guest User would be introduced, designed for temporary access with biometric data used only for immediate authentication without being stored in the database.

This tiered structure enhances the system and broadens accessibility while maintaining privacy and security standards. Each user is designed and tailored to help balance security measures with user-friendly access.

Table 1: User Types

User Type	Privileges	Access Level	Biometric Data
Admin	- Full system management - Audit log access - User account management - System settings configurations	Full Access	Administrators can manage and oversee the data encryption and storage policies
Base User	- Biometric data registration - Personal service access post-authentication - Biometric data management	Restricted to only personal authenticated areas	Base users can access encrypted storage and management of their own biometric data
Guest User	- Temporary biometric authentication - Access to public system areas without registration	Temporary access with limited areas	Guest users have the lowest tier and their data is immediately authenticated without storage

User Interface

The user interface has been developed using Python with Tkinter, a library that offers a practical design for helping create a responsive interface for smooth interaction with my system. The choice of Tkinter was made as it aligns with the objectives of the project as it provides a reliable UI that can quickly be prototyped and accessible for the system's users. The UI is designed to keep a simple layout with the interactive elements to allow users to engage with the system's functions - such as biometric data input, updates, and result notifications.

Although Tkinter was chosen for this system, the consideration to use a more traditional web-based interface - using HTML and CSS - was thought of, however, due to the sizing of this current project and time considerations the Tkinter system was deemed the most optimal decision (***See Appendices 1 through 1.5 'Future Web Implementation' for more information***). The future web version of the UI would use the advantages of modern technologies to help deliver an enhanced experience to the user, this version would also have greater scalable and secure features, while keeping the flexibility to incorporate more advanced features.

While the web based system would be a more ideal design, the decision to develop the system using a python GUI with Tkinter was chosen due to the time constraints, and limitations that were present. Tkinter is not as enhanced or flexible as the web based version however, it does provide a secure interface that allows for a richer variety of future implementations as it works in tandem with the Python developed back-end.

Choice of Language - Python

Ecosystem for Computing and Cryptography

Python has an extensive ecosystem of libraries that are geared towards scientific computing and data analysis, such as Numpy, which facilitate complex mathematical operations and data handling with ease. For Cryptographic operations, Python offers specific libraries - like the one chosen for this project - TenSEAL that integrate seamlessly with the ecosystem, creating tools to help implement encryption schemes such as CKKS, which is critical for Fully Homomorphic Encryption implementations *(Python for Data Analysis, 3E)*.

Wider Community Support

Python's popularity has been constantly growing and it's becoming one of the most popular programming languages. Its large community of developers means that it has extensive

documentation, tutorials and forums to help with creating and troubleshooting. This vast support makes it easier to help solve problems and find information on very specialised topics, such as Fully Homomorphic Encryption.

Simplicity and Readability of Code

Python has clear and concise syntax allowing it to be an easily readable code for beginners. This readability is crucial when working on projects such as the PPBA system as outlined throughout this report. “The Zen of Python”, which can be found in Python Enhancement Proposals, helps emphasise the importance of readability and simplicity when choosing a coding language and what Python's design and philosophy is.

Table 2: Code Choices

Category	Python	Java	C	C++
Ease of Use				
Library Support				
Performance				
Integration				
Development Speed				
Scalability				

Homomorphic Encryption

Why Fully Homomorphic Encryption was chosen?

Homomorphic Encryption (HE) has various different forms - Somewhat Homomorphic Encryption (SHE), which can only support a limited number of operations before the noise gets too large and the encrypted value is lost *(Michael Belland et al. , Somewhat homomorphic encryption 2017)*; Partial Homomorphic Encryption, which supports either addition or multiplication but you are unable to compute both. However, the current PPBA system needs the flexibility to perform a number of operations on the encrypted data to facilitate complex preprocessing and authentications. Due to this reason Fully Homomorphic Encryption stands out as the preferred encryption method.

Table 3: Encryption Method

Encryption Type	Multiplication	Addition	Noise Management	Real Number Support	Practical Cases
Partial HE	Limited	Limited	Simple	No	Voting Systems
Somewhat HE	Limited	Limited	Moderate	No	Encrypted Search
Fully HE	Unlimited	Unlimited	Complex	Yes	PPBA System

Fully Homomorphic Encryptions Unrestricted Operations:

Unlike SHE and PHE, Fully Homomorphic Encryption is not limited by the number or type of operations,as stated in the above table. This feature is essential for the creation of a PPBA system that will need to process biometric data through a series of complex computations.

Data Privacy Assurance:

FHE also ensures that the data remains encrypted throughout the entire computation process, which is a key requirement for preserving the privacy and sensitive biometric data. Partial and Somewhat HE do not offer the same level of assurance that FHE can offer the system, as they may require decryption at certain stages of computation.

Regulatory Compliance:

Creating an emphasis on data protection regulations like GDPR, Fully Homomorphic Encryption provides a robust framework to comply with legal standards, as data never needs to be decrypted, even during processing.

Implementation Libraries Chosen

TenSEAL

For this project TenSEAL was the chosen library. This is down to its focus when using homomorphic encryption, this is critical as it allows for me to perform computations on the data without compromising its privacy. Unlike other cryptographic libraries, TenSEAL is built upon Microsoft's SEAL creating a python-friendly API that will be optimised for the operations when handling the data. The library also offers support for operations like addition and multiplication on encrypted vectors - this aligns perfectly with the needs of the system as it handles processing for biometric data. These aspects made TenSEAL the ideal choice, as it provides all the necessary performance and security features needed to complete this project's outcomes.

Tkinter

For the development of the front-end for the system, Tkinter was chosen for its straightforward toolkit which allowed for rapid UI development. This library choice allows for a creation of a simple user interface that integrates seamlessly with the nature of the project. The integration of Tkinter with Python improves the systems coherence by linking the user interface directly with the operations, this integration is crucial for maintaining a consistent user experience across all the system's aspects.

Other Libraries

- **SqLite:** Manages a local SQLite database, critical for storing and retrieving the encrypted data and for comparison
- **Zlib + base64:** Used to compress the encrypted data and encode it into ASCII strings. This combination ensures the data remains compact across different operations
- **Logging:** Essential for tracking the systems operations for debugging and testing.
- **OS**: Used to manage the file paths and directories creating a robust system.

Database: SQLite

Database Functions

Figure 4 - Database Snippet (See Figure 4, References / Figures)

```
def initialize_database():
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS fingerprints (
                        user_id TEXT PRIMARY KEY,
                        encrypted_data BLOB
                    )''')
    conn.commit()
    conn.close()

def save_to_database(user_id, encrypted_data):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    compressed_data = zlib.compress(encrypted_data)
    cursor.execute("REPLACE INTO fingerprints (user_id, encrypted_data) VALUES (?, ?)",
                   (user_id, base64.b64encode(compressed_data)))
    conn.commit()
    conn.close()

def load_from_database(user_id):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT encrypted_data FROM fingerprints WHERE user_id = ?", (user_id,))
    row = cursor.fetchone()
    conn.close()
    if row:
        decompressed_data = zlib.decompress(base64.b64decode(row[0]))
        return decompressed_data
    return None
```

Initialisation

The system starts by initialising the database by setting up a ***'fingerprints'*** database table if one doesn't already exist. This table is designed to store user IDs and their corresponding encrypted biometric data. The SQL command used ensures that each user ID is unique and links to a specific string of encrypted data.

Data Storage

Once the database has been initialised the encrypted data is then compressed and base64 encoded to ensure compatibility and a reduced storage size - this is used using the ***'REPLACE INTO'*** command. This command updates existing entries if the user ID already exists, ensuring that the database contains the most updated version of the encrypted data.

Data Retrieval

When authentication is required during the system's process, the system retrieves the encrypted data by checking the database for the user ID. The retrieved data is then base64 decoded and decompressed, ready for the cryptographic operations to be performed.

Integrity and Security

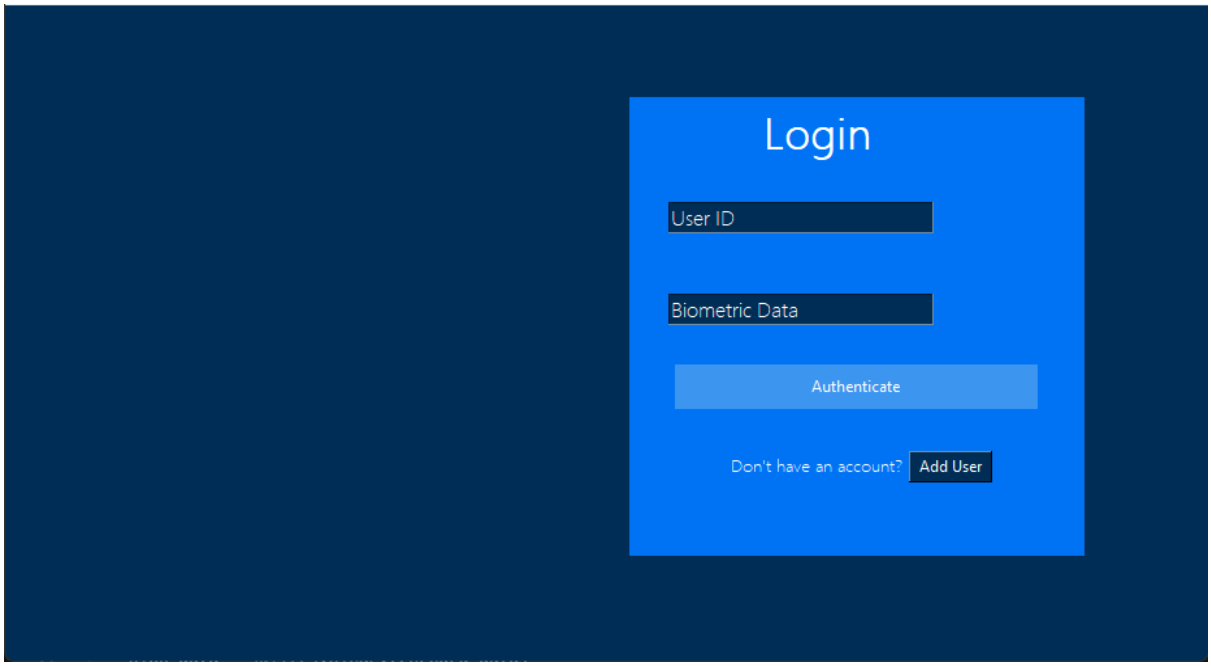
The database operations use error handling and use queries to help protect against attacks. This approach not only ensures integrity for the data but also enhances the security of the system.

Front-End: User Interaction

The front-end serves as the gateway between the user interface and the backbone of this system through the back-end processes. The visual design of the system's front-end has been chosen to give a simplistic look and functionality, to allow for the user interface to be as smooth and usable as possible. The simplicity also juxtaposes the cryptographic operations that are occuring on the backend of the system, which is the key focus and foundation to the system as a whole. This structure ensures that while the back-end functions are complex in nature, the front-end remains simple and user-friendly.

User Login

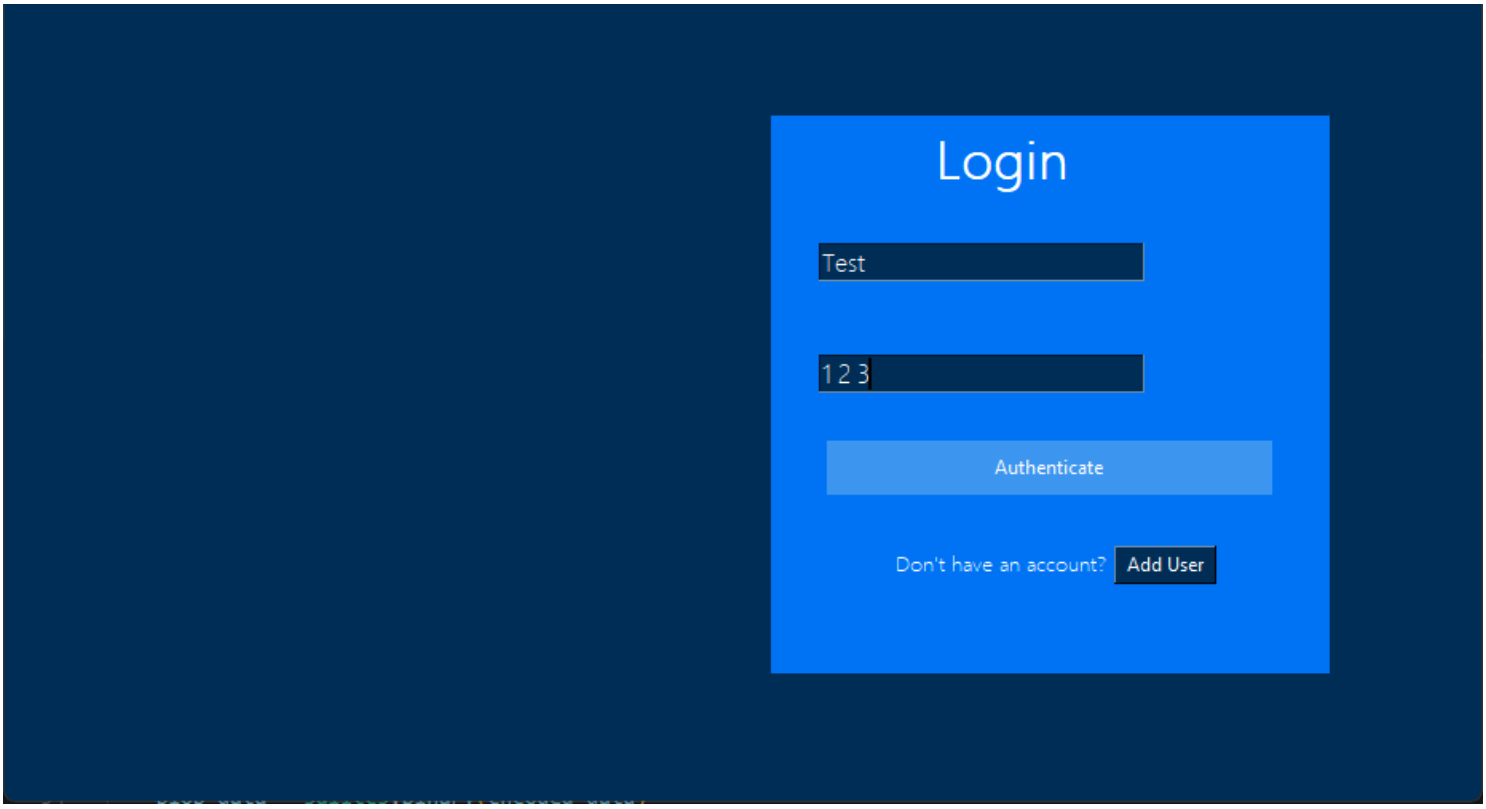
Figure 5 - Login Screen (See Figure 5, References / Figures)



The ‘Biometric Login’ screen is the door to the system that users first interact with when running the system. The background of the window is a Deep Blue colour with Electric Blue highlights to showcase the input boxes on the right side of the screen. The main deep blue colour is also used for the input boxes themselves to allow for them to stand out on the brighter frame to keep user interaction high and easy to follow.

The Interface includes a text box for the user’s ID and a separate box for the Biometric Data, at the bottom of the screen is a long brighter blue button titled ‘Authenticate’ which is used to compare the inputted data with data previously stored in the database. On the bottom right of the screen is a ‘Add User’ button that allows the user to open a new window to add their biometric data and user ID to the database ready to then Login and authenticate.

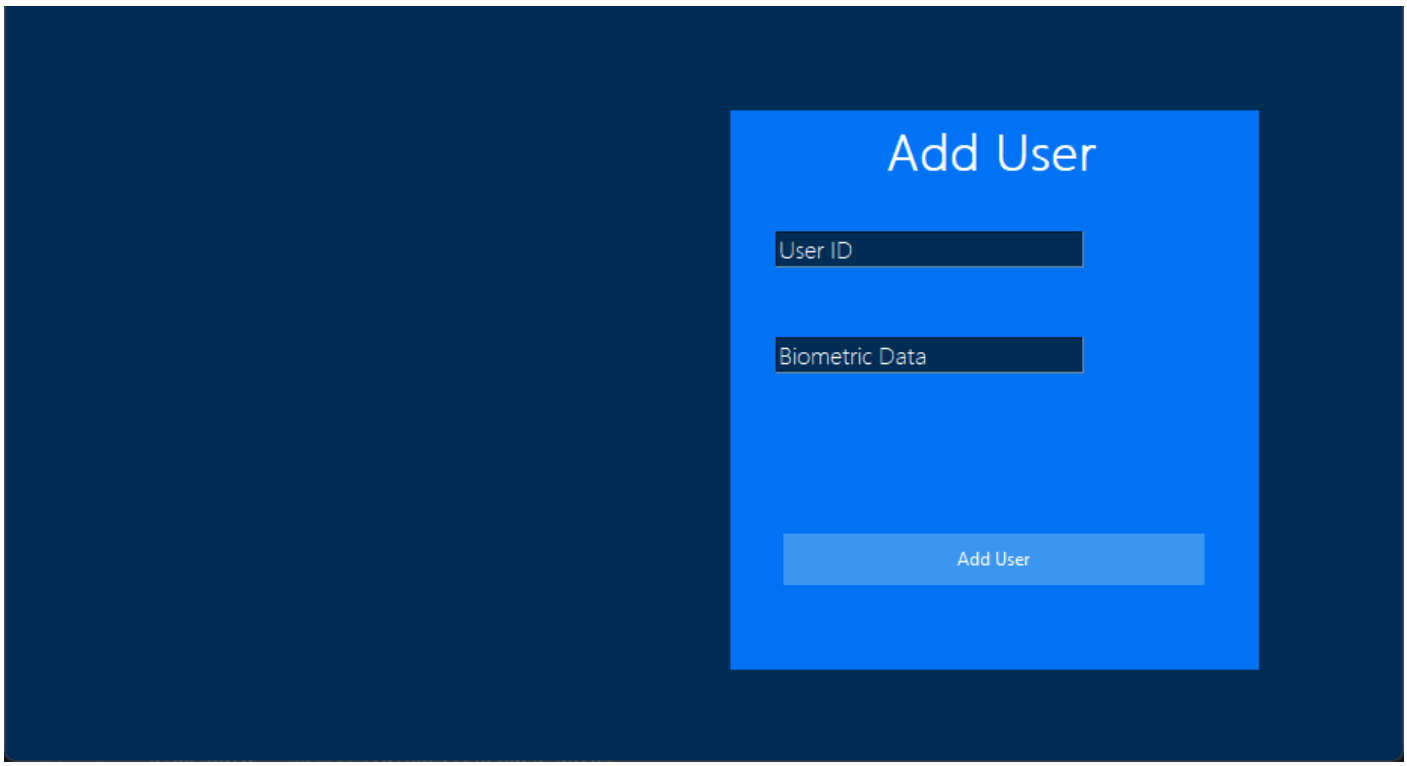
Figure 6 - Login Screen P2 (See Figure 6, References / Figures)



When clicking on the input buttons the stored text ‘User ID’ and ‘Biometric Data’ is removed, so that the user is able to input their own data, if no data is in the input box then the stored text will reappear to signify where the user is meant to input the data.

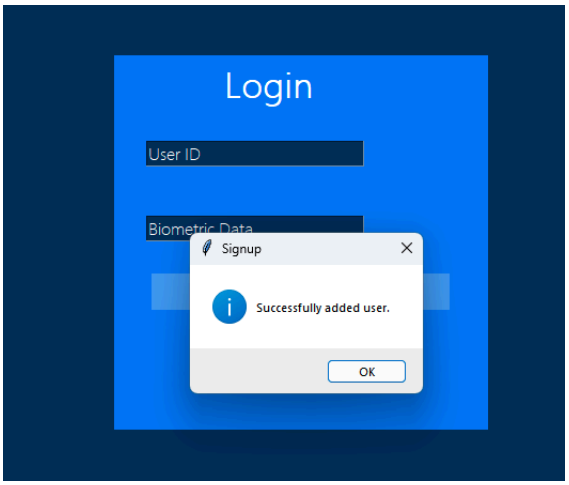
Add User

Figure 7 - Add User Screen (See Figure 7, References / Figures)



The 'Add User' window has the same colour scheme as the Login page to keep both the pages consistent and straightforward for users to follow. Users are prompted to enter a User ID and Biometric data before submitting the data to the database to be encrypted and stored through the 'Add User' button on the bottom of the frame.

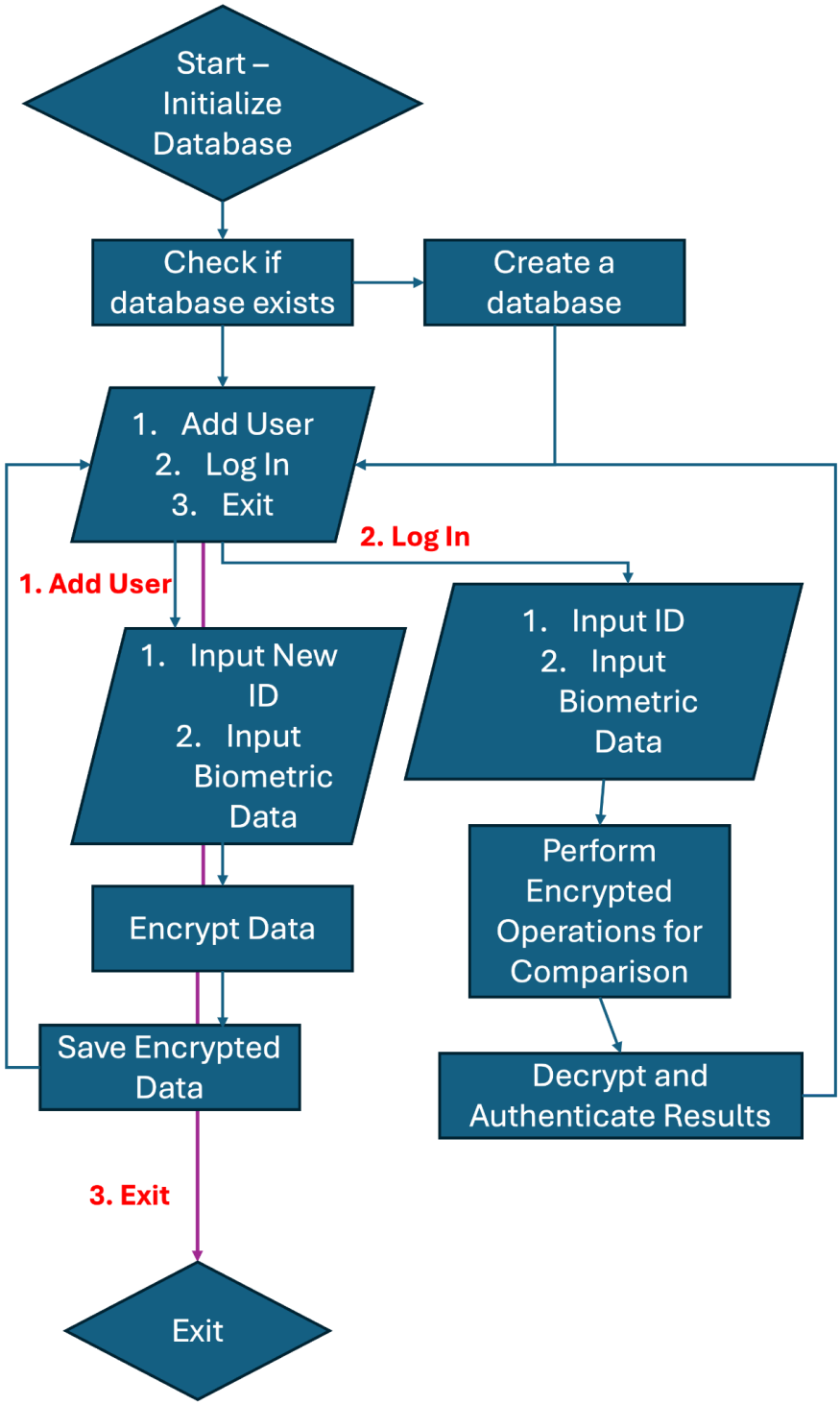
Figure 8 - Add User Screen P2 (See Figure 8, References / Figures)



When clicking the 'Add User' button the user is prompted with a pop up signifying if the data is successfully added or if an error has occurred. Once the 'Add User' button is pressed the Add User window will automatically close to allow the user to then take their new data and authenticate.

Back-end Design: Flowchart

Figure 9 - Back-end Flowchart (See Figure 9, References / Figures)



The back-end is the key aspect of the Privacy-Preserving Biometric Authentication (PPBA) system in order to be able to handle sensitive biometric data securely through several different components: Encryption, data storage, and data comparison. These components are critical for allowing the system to maintain integrity and confidentiality of the users data while providing a robust system for authentication.

Encrypting and Storing Data

The system utilises a fully homomorphic encryption scheme. This approach allows computations to be performed on the inputted data, ensuring that the data remains secure throughout the systems life cycle. This process converts biometric data into a format that can be securely stored and processed without exposing the sensitive plain text.

Once encrypted, the biometric data is compressed using the ***'zlib'*** library to reduce the size, making storage more efficient and allowing for more data to be stored in the database when increasing the system's application. The compressed data is then encoded into a base64 format to ensure it can be handled as a text string, this is crucial for storage in databases that may not support binary formats. This process is handled by the ***'save_to_database'*** function, which inserts or updates the inputs into the SQLite database.

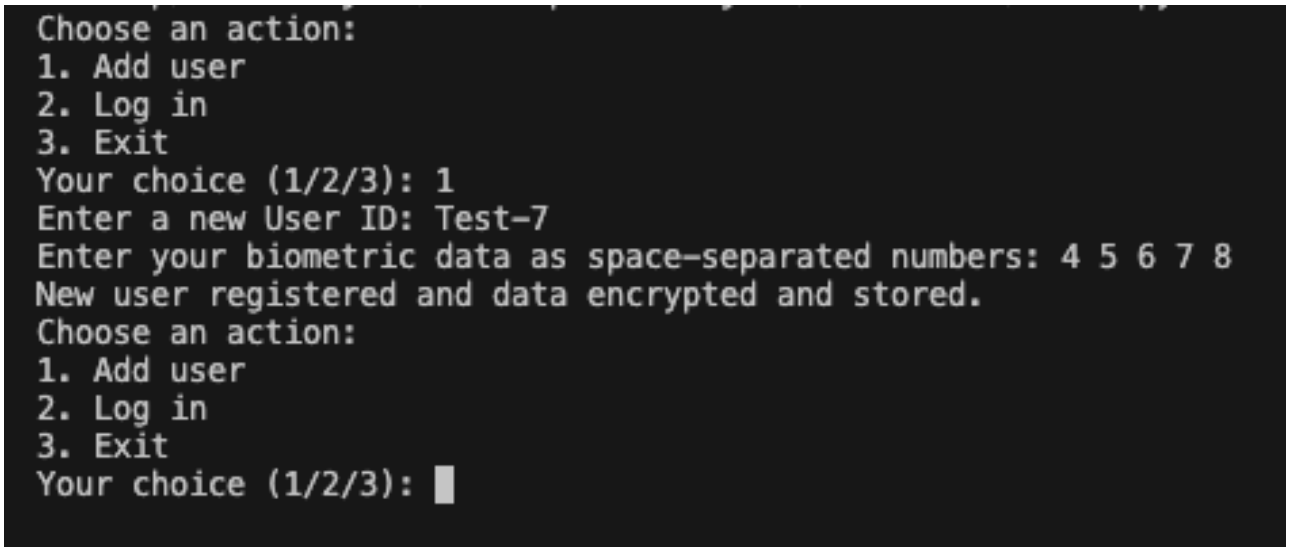
Comparison and Authentication of Data

The ***'load_from_database'*** function retrieves the encrypted data, which is then base64 decoded and decompressed. The system then uses a comparison functionality, this step is critical for authentication purposes. Encrypted data vectors are compared to verify if the given biometric data matches the stored data. This comparison is done under the encrypted data using the Fully Homomorphic Encryption process ensuring that the comparison process does not compromise the security and privacy aspects of the system. However, the system makes sure that this only occurs during the authentication phase. This structure allows the system to perform authentication checks without ever exposing the original plain text.

Terminal access and CLI

When accessing the code if the front-end isn't available, terminal access is provided through the use of a command line interface (CLI), designed to facilitate easy interaction with the system.

Figure 10 - CLI Interaction (See Figure 10, References / Figures)



The CLI allows the system to perform various tasks such as adding users, updating data, and conducting the authentication checks. The CLI is built using the chosen language for the system - Python - this provides a simple way to interact with the key backend processes. It prompts the user to choose one of the 3 actions and, based on the user's input, calls the appropriate functions to handle encryption, storage, and comparison of the data.

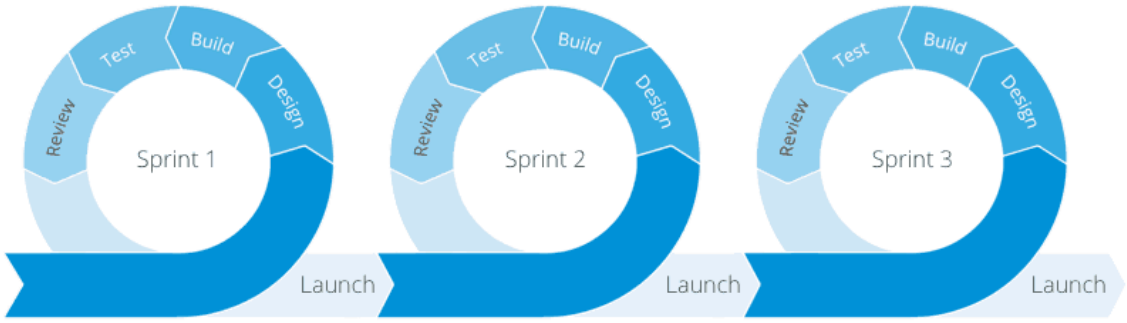
The design of the CLI is focused on simplicity and usability, providing clear instructions and feedback to the user, ensuring that the system is accessible. The error handling is robust, with the system capable of handling and logging errors, which aids in troubleshooting and limit testing keeping the system reliable and stable.

Development Lifecycle + Iterations

Agile Method Adaptation

The Agile development cycle is an adaptive planning system that allows for evolutionary development and flexibility. Within this project, Agiles principles were perfectly tailored to help the specific needs of developing a complex encryption system. Having regular meetings ensured alignment, while the sprints focused on incremental delivery of key functional components. This flexibility allowed for pivots in response to the feedback given from peers and my supervisor - ensuring that each iteration purposefully navigated to the desired endpoint.

Figure 11 - Agile Methodology (See Figure 11, References / Figures)



Reviews and Meetings

Throughout the development process, peer reviews and meetings were held regularly, these functioned as milestones for the validation and reflection of the different iterations of the project. These engagements helped dive into the project's status addressing the immediate technical aspects that needed review. By weekly presenting my progress, I was able to align the development trajectory with both my supervisors expertise and the needs of the system. These sessions became a pillar for the improvement and development of the project allowing it to propel forward with ease.

Feedback and Back-end Iterations

The Agiles framework's biggest strength is within its iterative nature. Each development cycle turns thorough reviews and feedback into actionable insights that helped improve the system. The

process allowed for every piece of feedback to be placed into the system, resulting in a progressively more developed and user aligned product.

First Iteration

Figure 12 First Iteration Snippet (See Appendices - First Iteration for further aspects of the code)

```
input_data = input("Do you want to input fingerprint data? (yes/no): ")
if input_data.lower() == "yes":
    fingerprint_data = list(map(float, input("Input Fingerprint Data (numerical values separated by space): ").split()))
    fingerprint_data = preprocess_fingerprint_data(fingerprint_data)

    encrypt = input("Do you want to encrypt? (yes/no): ")
    if encrypt.lower() == "yes":
        context = create_context()
        encrypted_data = encrypt_biometric_data(fingerprint_data, context)

        user_id = input("Enter user ID for storing the data: ")
        write_data(f'{user_id}_encrypted_fingerprint.txt', encrypted_data)
```

The initial iteration of the system was presented in the First Deliverable of this project and while providing a basic version of the system - thorough testing revealed that the encryption using FHE was implemented successfully, with the system efficiently encrypting data - as indicated by FT-01 (***List of Tables: Test Cases - Iteration 1).*** However, test ST-01 suggested that while the encryption was successful, the operations that were being performed and the keys used could've been a lot more secure.

Table 3: Test Cases - Iteration 1, FT-01

ST-01	Security for encryption method	No vulnerability in the encryption	Encryption does occur, however, the operations performed and keys used are not secure.	

The decryption tests - FT-02 - showcased that the system could change the encrypted data back to its original, however, the new text did not match the originally inputted data - indicating that there was an issue with the reversal of the operations. The CLI performed well, handling inputs efficiently as seen in FT-03. Performance tests - PT-01 and PT-02 - also indicated that both the encryption and decryption were executed speedily, allowing for the system to meet the expectations of the current system design.

Table 4: Test Cases - Iteration 1, FT-03

FT-03	Command-line interface user interaction test	CLI handles user inputs correctly	CLI handles users inputs easily and efficiently	

However, certain critical areas need improvement for the next iteration of the system. This version currently doesn't support the use of a database, meaning that no data comparison or authentication could be performed. The DT - Database Tests - tests were also deemed as an error in this iteration, as the database functionalities have not been incorporated into this current iteration.

Second Iteration: Improvements and Testing

Figure 13 Second Iteration Snippet (See Appendices - Second Iteration for further aspects of the code)

```
def encrypt_biometric_data(data, context):
    logging.debug("Encrypting biometric data")
    encrypted_data = ts.ckks_vector(context, data)
    return encrypted_data.serialize()

def decrypt_data(encrypted_data, context):
    logging.debug("Decrypting data")
    #Decrypts the given encrypted data using the provided TenSEAL context.
    encrypted_vector = ts.lazy_ckks_vector_from(encrypted_data)
    encrypted_vector.link_context(context)
    decrypted_data = encrypted_vector.decrypt()
    return [round(num, 2) for num in decrypted_data]
```

After the first iteration and testing, changes were made and testing was once again performed. The first encryption test (FT-01), despite successfully encrypting the data, didn't fully perform to the systems outcomes. Unlike the first iteration, the second failed to engage the full potential of homomorphic operations indicating that the system was not yet performing as intended.

Table 5: Test Cases - Iteration 2, FT-01

FT-01	Encryption of data using FHE	Data is fully encrypted and not visible	Data is encrypted however, there is a lack of homomorphic operations applied to the data, current system uses simplistic arithmetic	

The decryption process (FT-02) continued to show its strength - consistently transforming the data back to its plaintext form - demonstrating an improvement in data integrity and consistency within the iterations. This continued with the CLI (FT-03) as this testing also remained robust, with the user interaction still remaining smooth and efficient.

Contrastingly, the initial security test (ST-01) indicated that while the encryption is performing and staying secure, the integration of homomorphic operations is lacking compared to the first iteration. Authentication tests revealed a significant challenge, the current iteration of the system fails to match encrypted data with stored data for authentication. Although the database storage (DT-01) and retrieval (DT-02) are successful the comparison and authentication were not functioning properly.

Table 6: Test Cases - Iteration 2, ST-01

ST-01	Security for encryption method	No vulnerability in the encryption	The encryption is performed to a secure standard however due to the lack of homomorphic operations makes the code not completely secure	

Third Iteration: Final Changes

After the testing and changes from Iteration 1 & 2, the systems Encryption (FT-01) and Decryption (FT-02) processes were executed with further precision, highlighting the system's progress in handling the cryptographic functions that are needed for the system to function. The FHE operations have improved to reach a level of efficiency that was not fully optimised in previous iterations.

Table 7: Test Cases - Iteration 3, FT-01 & FT-02

FT-01	Encryption of data using FHE	Data is fully encrypted and not visible	Data is encrypted fully optimising the FHE operations	
FT-02	Decrypt data back to its original form	Data returns to its original state	Decryption and authentication works as attended	

Figure 14 Final Iteration Snippet (See Appendices - Final Iteration for further aspects code)

```
def perform_encrypted_comparison(encrypted_data1, encrypted_data2, context):
    vector1 = ts.ckks_vector_from(context, encrypted_data1)
    vector2 = ts.ckks_vector_from(context, encrypted_data2)

    difference_vector = vector1 - vector2
    sum_vector = vector1 + vector2
    product_vector = vector1 * vector2

    decrypted_differences = difference_vector.decrypt()
    decrypted_sums = sum_vector.decrypt()
    decrypted_products = product_vector.decrypt()

    logging.debug(f"Decrypted differences: {decrypted_differences}")
    logging.debug(f"Decrypted sums: {decrypted_sums}")
    logging.debug(f"Decrypted products: {decrypted_products}")

    return decrypted_differences, decrypted_sums, decrypted_products
```

The system has shown no vulnerabilities in regards to the security testing (ST-01) which helps show an improvement from previous iterations in regards to establishing the systems robustness. Performance testing for both encryption and decryption (PT-01, PT-02) maintained the momentum, with operations being performed swiftly and accurately.

Table 8: Test Cases - Iteration 3, PT-01 & PT-02

PT-01	Performance speed for encryption	Encryption completes with acceptable time	Encryption is done in a speedily manner	
PT-02	Performance speed for decryption	Decryption completes with acceptable time	decryption is done in a speedily manner	

Database interactions (DT-01 - DT-04) also underwent positive improvements between the previous iterations, now performing speedier and error-free data transactions. The storage and retrieval of the users data (DT-01 and DT-02) demonstrated the capacity for handling the data both securely and efficiently.

Implementation of the system

User Interaction - GUI

Design and Functionality

The system's front end was implemented using Tkinter because of its simplicity and effectiveness when designing the needed aspects for this system. Tkinter allows for the development of a user focused interface that can interact with the python functions needed for the back-end to function. The GUI is structured to include input areas for user IDs and Biometric data, and buttons to allow for the user to submit the data for storage or authentication.

Figure 15 - GUI Structure (See Figure 15, References / Figures)

```
root = Tk()
root.title('Biometric Login')
root.geometry('925x500+300+200')
root.configure(bg='#fff')
root.resizable(False, False)
```

This section of the system helps initialise the main window of the application, giving it a size, title, and background colour, to provide the best layout for the users.

User Registration and authentication

The front-end has specific functionalities for the user to input data needed for authentication and registration - these are critical implementations for the systems operations. The registration interface allows new users to enter their unique ID and biometric data, which is then encrypted and stored securely in the SQLite database. The authentication (Login) interface allows for the user to enter their ID and data to access the system - by taking the inputted data and authenticating it against previously stored data. Both of these functionalities are crucial for the data encryption before any processing, maintaining the privacy and security of the biometric data.

Figure 16 - Sign in Function (See Figure 16, References / Figures)

```
def signin():
    username = user.get()
    data = [float(x) for x in code.get().split()]
    encrypted_input_data = ts.ckks_vector(context, data).serialize()
    encrypted_stored_data = load_from_database(username)
    if encrypted_stored_data:
        comparison_result, sums, products = perform_encrypted_comparison(encrypted_input_data, encrypted_stored_data, context)
        if comparison_result:
            messagebox.showinfo('Login Success', 'Biometric authentication successful.')
        else:
            messagebox.showerror('Invalid', 'Biometric authentication failed.')
            logging.error(f"Verification failed for user {username}")
    else:
        messagebox.showerror('Invalid', 'No user found with this ID.')
```

This function is used for the login process, where the user inputs are encrypted and compared against stored data to authenticate a user. It highlights the privacy-preserving nature of the system and ensures that the system operates as needed.

Figure 17 - Input Placement (See Figure 17, References / Figures)

```
user_add.bind('<FocusIn>', lambda e: user_add.delete(0, END) if user_add.get() == 'User ID' else None)
user_add.bind('<FocusOut>', lambda e: user_add.insert(0, 'User ID') if not user_add.get() else None)

code_add = Entry(frame, width=25, fg='#FFFFFF', border=1, bg='#003366', font=('Microsoft YaHei UI Light', 11))
code_add.place(x=30, y=150)
code_add.insert(0, 'Biometric Data')
code_add.bind('<FocusIn>', lambda e: code_add.delete(0, END) if code_add.get() == 'Biometric Data' else None)
code_add.bind('<FocusOut>', lambda e: code_add.insert(0, 'Biometric Data') if not code_add.get() else None)
```

The ‘FocusIn’ and ‘FocusOut’ commands allow for the stored signifier text that is placed in the input boxes to be removed when the user clicks on the boxes to input their own data. This allows for the user to see where their data is needed to be inputted but also to have the box be empty when the user needs to input their data. Having these functions gives the user better interaction with the system as the front-end becomes more clear and concise with what goes where.

Feedback and Error Handling

The GUI is also tasked to provide visual feedback to users based on the success or failure of the authentication process. This feedback is crucial for keeping the user experience high and communicating the systems responses to the users actions.

Figure 18 - Error handling (See Figure 18, References / Figures)

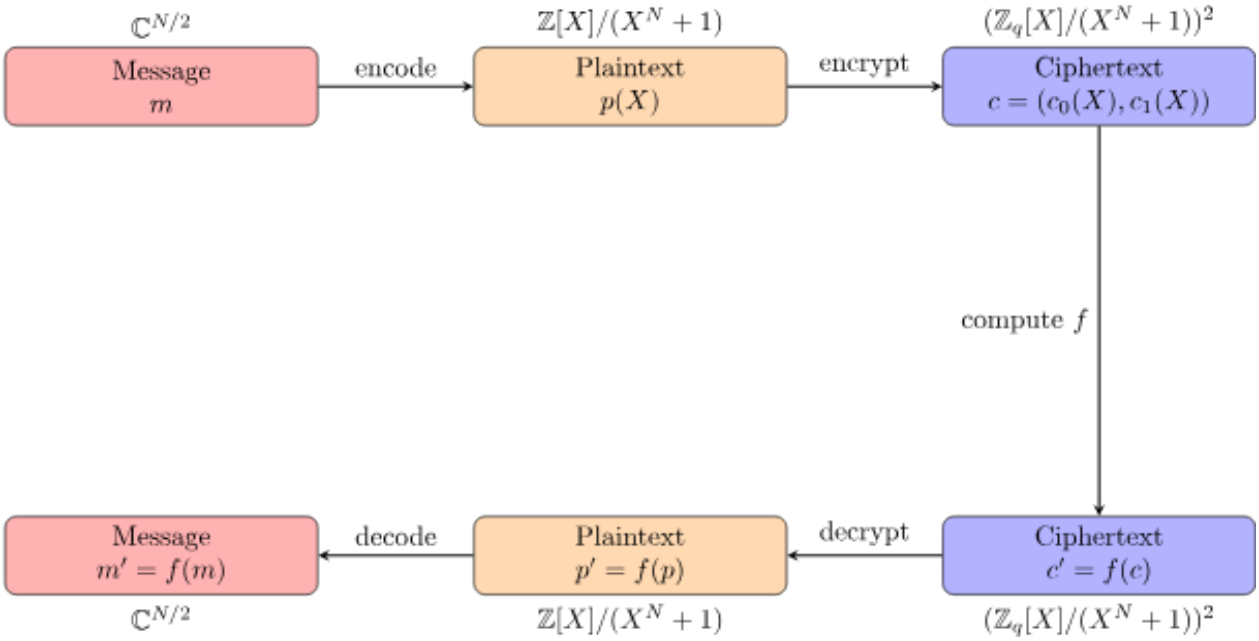
```
        messagebox.showinfo('Login Success', 'Biometric authentication successful.')
    else:
        messagebox.showerror('Invalid', 'Biometric authentication failed.')
```

Encryption Methodology

CKKS Scheme

CKKS offers a practical approach to FHE that helps align with the requirements needed for the system. This is because it allows for approximate arithmetic operations to be used on the encrypted real or complex numbers, which is particularly suitable when it comes to using biometric data that includes noise and does not always require the exact arithmetic operations.

Figure 19 - CKKS Scheme (See Figure 19, References / Figures)



Mathematical Representation in the code

Initialisation of Encryption Parameters

The creation and management of the encryption context is crucial for setting the groundwork for all of the operations performed. Within the system, this is handled by initialising the ***'context'*** with specified polynomial modulus degree and coefficient mod bit sizes, which help with defining the complexity and security level of the encryption scheme.

Figure 20 - Create Keys (See Figure 20, References / Figures)

```
def create_context_and_keys():
    if os.path.exists(KEYS_FILE_PATH):
        with open(KEYS_FILE_PATH, 'rb') as key_file:
            context = ts.context_from(key_file.read())
            logging.info("Encryption context loaded from existing keys.")
            return context
    else:
        context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree=8192, coeff_mod_bit_sizes=[60, 40, 40, 60])
        context.generate_galois_keys()
        context.global_scale = 2**40
        with open(KEYS_FILE_PATH, 'wb') as key_file:
            key_file.write(context.serialize(save_secret_key=True))
        logging.info("New encryption context and keys generated.")
        return context

context = create_context_and_keys()
initialize_database()
```

This code initialises the Fully Homomorphic Encryption context using a CKKS scheme which allows for arithmetic operations on encrypted data. The ***'poly_modulus_degree'*** and ***'coeff_mod_bit_sizes'*** are directly used for the encryptions security and computation operations.

Serialisation and Deserialisation of Encrypted Data

Operating biometric data means that the system has to have secure transmission and storage of the data once it is encrypted, this is done through the use of CKKS vectors and serialisation. The process of serialisation allows for the encrypted data to be transformed into a format that is able to be stored and transmitted securely, while the deserialisation converts it back into a format that can be used for operations.

Figure 21 - Serialisation (See Figure 20, References / Figures)

```
data = [float(x) for x in code.get().split()]
encrypted_input_data = ts.ckks_vector(context, data).serialize()
encrypted_stored_data = load_from_database(username)
```

Comparison + Authentication

Privacy-Preserving Techniques

The comparison functions are crucial for the system as it is the key for verifying the identity of the user by matching their biometric data against the encrypted data stored in the system.

Figure 22 - Encrypted Operations (See Figure 20, References / Figures)

```python
def perform_encrypted_comparison(encrypted_data1, encrypted_data2, context):
    vector1 = ts.ckks_vector_from(context, encrypted_data1)
    vector2 = ts.ckks_vector_from(context, encrypted_data2)

    # Check if the sizes of the vectors match using the size() method
    if vector1.size() != vector2.size():
        logging.warning("Attempted to compare vectors of different sizes.")
        return False, [], []  # Automatically fail authentication

    difference_vector = vector1 - vector2
    sum_vector = vector1 + vector2
    product_vector = vector1 * vector2

    decrypted_differences = difference_vector.decrypt()
    decrypted_sums = sum_vector.decrypt()
    decrypted_products = product_vector.decrypt()

    logging.debug(f"Decrypted differences: {decrypted_differences}")
    logging.debug(f"Decrypted sums: {decrypted_sums}")
    logging.debug(f"Decrypted products: {decrypted_products}")

    return all(abs(diff) < 0.1 for diff in decrypted_differences), decrypted_sums, decrypted_products
```

Performing Encrypted Comparisons

One of the core functionalities of this system is the functions to perform encrypted comparisons and authentications on data without revealing the plaintext, this allows for biometric verification while preserving privacy. This function takes two pieces of the encrypted data along with the context which contains necessary keys and parameters, this context is a crucial element as it holds the environment where the cryptographic operations will occur. The function performs three operations using FHE these being addition, multiplication and subtraction (difference_vector, sum_vector and product_vector) these operations are done to authenticate the inputted data with the stored data, without revealing the plaintext.

User Authentication

Figure 16 - Sign In Function (See Figure 16, References / Figures)

```
def signin():
    username = user.get()
    data = [float(x) for x in code.get().split()]
    encrypted_input_data = ts.ckks_vector(context, data).serialize()
    encrypted_stored_data = load_from_database(username)
    if encrypted_stored_data:
        comparison_result, sums, products = perform_encrypted_comparison(encrypted_input_data, encrypted_stored_data, context)
        if comparison_result:
            messagebox.showinfo('Login Success', 'Biometric authentication successful.')
        else:
            messagebox.showerror('Invalid', 'Biometric authentication failed.')
            logging.error(f"Verification failed for user {username}")
    else:
        messagebox.showerror('Invalid', 'No user found with this ID.')
```

The system's UI allows for a streamlined and straightforward interaction for the inputs of User IDs and biometric data, which is then encrypted and stored or compared against already stored data. Successful authentication results are then sent back to the user via a message box pop up window, enhancing the users experience with straightforward responses.

Testing Methodology

During the final testing process for the completed version of the system, a meticulous and structured methodology was ensured to make sure that each function was intended as it should, to allow for the highest standards of security and integrity throughout all the system's functions. The primary areas of focus were the functionality, security, performance, and usability, the testing was conducted through various strategies such as unit testing, integration testing, security testing, and usability testing (See appendices Test Cases - Final Iteration).

Back-End Functions

Database Initialisation (TC-001)

Initial testing was concentrated on the establishment of the systems repository. This is done through the systems SQLite database, ‘prints_Main.db’ and the ‘fingerprints’ table is crucial. This testing helped validate the back-ends effectiveness and readiness for managing the biometric data, confirming the correct structuring of the data’s storage.

Table 9: Test Cases - Final Iteration, TC-001

TC-001	Database Initialisation Test	Run the System	The database and table should exist with the correct configuration	System Runs	
		Check if the SQLite database file ***'prints-Main.db'***		Database file is created	
		Verify if the table ***'fingerprints'*** os created		Fingerprint File is created	

Data Encryption and Storage (TC-002)

After the previous iterations and testing the final iteration and application of the system underwent another set of testing revolving around the encryption and storage of the biometric data and other functions, this was done through the utilisation of 'save_to_database' function, the testing ensuring that this function verified the encryptions process efficeincy, ensuring that the biometric data is stored securely.

Table 10: Test Cases - Final Iteration, TC-002

TC-002	Save Encrypted data to the database	Input encrypted data for the user	Data is correctly compressed, encoded, and saved within the database	User gets to input data and it successfully encrypted	
		Calls ***'save_to_database'*** with the users ID and encrypted data		Users data is saved to the database through the ***'save_to_database'*** function	
		Check the database to see if data is saved		Database is securely stored	

Retrieval Accuracy and Integrity (TC-003)

To finish the back-end testing, the retrieval process was tested to see if the data was accurate and maintaining integrity throughout the functions. This was done through the 'load_from_database' function, this step authenticated the system's capacity to retrieve, decompress and decode the previously stored data without compromising the integrity.

Table 11: Test Cases - Final Iteration, TC-003

TC-003	Load Data from the database	When 'logging in' make sure data is recalled from database	The functions returns the correct data	During authentication functions only run if matching data is found stored in the database	
		Call ***'load_from_database'*** to see if data can be called		***'load_from_database'*** is used correctly to retrieve data	
		Verify the data is correctly decompressed and decoded		Data is being decompressed and decoded when leaving the database	

Front-End Usability

User Interface Interaction (TC-006)

In order to test the GUI functionality direct interaction tests were performed - trying various inputs (User IDs and Biometric Data) to ensure that the front-end could handle different sets of data. This examination confirmed the systems responsiveness to the users inputs, such as registering new user IDs and accessing the stored data.

Table 12: Test Cases - Final Iteration, TC-006

TC-006	GUI Functionality Test	Start the application	GUI accepts input, and the application processes and responds to authentication requests correctly	Application starts	
		Use the GUI to input a new user		Successfully inputs a new user	
		Attempt to login using stored data		Login successful	

Validation Testing

Functional Integrity (FT-001)

To ensure that all aspects of the system functioned properly and to affirm that the system's reliability and readiness was optimal, the functions were validated for their overall integrity. These tests were performed alongside the same criteria used for the previous iterations to ensure consistency and to provide insight to the systems Fully Homomorphic Encryption capability, while ensuring that the data could be returned to its original state.

Security Assurance (ST-001)

Testing to make sure that the security of the system was paramount, this was done through focusing on the encryption process to detect if any potential vulnerabilities occurred when testing the system. The results helped affirm that the encryption process correctly performed secure operations that allowed the system to uphold its requirements and keep the sensitive biometric data confidential throughout its operations.

Efficiency & Performance (PT-001)

The system's efficiency was then tested through the speed at which the main operations and functions were performed - Encryption, comparison and authentication. The testing helped validate that all key functions of the system accurately worked and outputted within the correct time frame allowing for the system to deliver its operations at a user-friendly speed.

Error Handling (UT-001)

Although the system currently uses a front-end in order to access and handle all the user interactions, a CLI back-end usability and error handling test was conducted to further test the robustness of the system, in various user interaction stages. This validation helped check to see if the CLI was free from errors to ensure that the user's interaction with the system was smooth and easy to use.

Robustness (CT-001)

The final testing done on the system was to assess the authentication processes robustness when outputting various scenarios. This involved verifying the systems ability to accurately authenticate users based on inputted and stored data. The results throughout this testing performed as hoped, with no errors, ensuring that the system operates as expected.

Use Cases (CT-002)

During the testing of the robustness and accuracy of the authentication functions, a short use case application was presented and tested on the system. These use cases tried various combinations regarding, wrong biometric data lengths, mismatching user ids and data, or trying to authenticate unstored data. These use cases helped give a visual representation to the code to ensure that various scenarios would be accounted for and tested against the robustness of the system ***(See appendices: 6. Use Case Tests).***

Table 13: Test Cases - Final Iteration, CT-002

CT-002	Use Cases	Stored: ID = Admin Data = 1 2 3 Input: ID = Admin Data = 1 2 4	Authorisation Failed	Window pop up with the notification that the authorisation has failed	
		Stored: ID = Admin Data = 1 2 3 Input: ID = Admin Data = 1 2 3	Authorisation Successful	Window pop up with the notification that the authorisation was successful	

The testing of the final iteration of the Privacy-Preserving Biometric Authentication system highlights its robustness when securely managing the sensitive biometric data. Through these series of meticulously conducted tests, both the back-ends functionality and front-ends usability has shown a high efficiency of security and user accessibility. This consistent testing has allowed for the system to perform as intended ensuring it meets the cryptographic standards while meeting a smooth user experience.

Critical Evaluation

The Privacy-Preserving Biometric Authentication system showcases an application of Fully Homomorphic Encryption in order to secure biometric data, highlighting the critical privacy and security concerns showcased in typical biometric systems. This critical evaluation will explore the strengths and limitations of the current system through its developmental iterations and practical applications.

System Architecture & Implementation Review

The final systems design is built on top of a framework that integrates FHE through the use of the TenSEAL library and CKKS encryption scheme to ensure that biometric data remains encrypted throughout all processes. This approach was chosen as it not only allows the data to remain secure against unauthorised access but also keeps the users data private by allowing for computations to be performed directly onto the encrypted data. The back-end of the system is the main functionality and has been designed and tested to handle complex cryptographic operations, with logging and error handling to maintain integrity and fix issues.

Design Performance & Challenges

Despite its developed back-end, the current system employs a simplistic front-end design created using Tkinter, this was done due to time constraints and the need to prioritise functionality over the aesthetic. While the design is straightforward and simplistic it still ensures that the user interaction is easily understandable for all functions, especially in the user-facing applications - like authentication - where the users engagement with the system is paramount for the overall experience of the system. A further, more complex and sophisticated front-end was designed and planned out, if time allowed, in order to help showcase how a fully fledged front-end would communicate and operate with the back-end in order to provide the best user experience and interaction with the system.

The key challenge when developing this system was the computational overhead that was being introduced by the integration of Fully Homomorphic Encryption. While FHE did help provide increased security benefits, it also demanded more processing power, which in the earlier iterations impacted performance andscalability. As evident in the initial iterations of the system this challenge caused issues with the authentication and comparison processes where they were not consistently accurate, thankfully, after numerous tests and changes this challenge was resolved and in the final testing of the system all processes ran with little to no errors.

Iterative Testing

A crucial aspect when designing the final system for this project was the use of iterative development and testing. Each major iteration of the system was tested against the same string of tests in order to get feedback and information regarding issues and improvements from previous tests. Each iteration addressed specific challenges from previous versions, such as encryption efficiency to ensure comparisons were accurate. These improvements through the various iterations were vital for helping the system evolve and to ensure that the system's capabilities met the standards of the project.

Future Advancements

Computational Efficiency

A key focus for future implementations is to focus on enhancing the computational efficiency even further. This enhancement would be crucial for allowing for the system to implement real-time biometric authentication, this improvement would implement the system into various new sectors, including finance, and healthcare. Current research is being aimed at trying to reduce the latency issues associated with FHE, making it more accessible. For example, significant improvements have been seen in the work of Costache and Smart (2016), who delve into the various optimisation techniques that could be applied to FHE.

Integration with Mobile and IoT

Along with a more sophisticated front-end - as discussed in previous sections - the system would benefit with research and future implementation into mobile and IoT devices. This development would help improve the system's range of authentication systems, making it once again more accessible and versatile to the public. This is further shown with current research, aiming to develop lightweight cryptographic models.

Scalable Models

In future implementation I would explore a more scalable deployment that can accommodate various sizes and types of biometric data and various organisations. These implementations would aim to deploy the PPBA system efficiently in both small-scale businesses and larger enterprises. In order to give the system this scalability I would potentially use cloud-based services in order to allow for the system to easily manage the increase in data. Bowers et al. (2017) provides an insight into how cryptographic operations can be scaled and adapted for cloud-based systems, this would help provide a black-box for giving the current system scalability.

User-Centric Security

Allowing the system to become more user-centric would also be a key future addition, this would give the user more control and transparency over where, and how their information is being stored. These features could include logs that are visible to the users, or an interface that allows users to see how their data is actually being protected and used. Unfortunately, due to the time constraints of the project specific user types and access levels were not able to be implemented fully, in a future implementation in order to help further improve the user-centric aspect of the system I would incorporate the user type system into the code to allow for specific users to have access to various different abilities of the system.

System Objective Review

The system's objectives were aimed at ensuring the user's data is securely and efficiently verified while maintaining the highest standards of privacy.

System Operations

The core foundation of this system is the registration, encryption, secure storage, and privacy-preserving authentication of the user's biometric data. These functions are essential for protecting the data from unauthorised access and ensuring that privacy is maintained throughout all steps.

Encryption and Storage

After registration of the data, the system uses FHE to encrypt the biometric data, allowing for the complex computations to be performed while maintaining its confidentiality. This encrypted data is then stored in a database using SQLite, ensuring that even in the event of unauthorised access the data will remain confidential.

Privacy-Preserving Authentication

Alll authentication done with the system is done through operations performed on the encrypted data. This ensures that the user's data remains completely secure throughout these processes. Advanced encryption methods are also used in order to ensure that biometric data is handled securely, this provides a reliable solution for organisations that require high levels of security and privacy. The system's architecture also supports these operations without compromising on performance, allowing the system to remain efficient even as it computes complex cryptographic processes.

Conclusion

The PPBA system successfully integrated Fully Homomorphic Encryption to ensure that biometric data remains encrypted throughout its processing, effectively addressing the challenges of securing sensitive information and maintaining user privacy. This conclusion will reflect on the achievements, challenges, and future implications of the system as detailed throughout the previous sections of this report.

The Privacy-Preserving Biometric Authentication system has successfully integrated the use of Fully Homomorphic Encryption to allow for biometric data to remain encrypted throughout its processing lifecycle. The successful implementation of this scheme is the key feature of the system and helps demonstrate the systems capability to safeguard against the rising threats and unauthorised access within the cyberworld. While the system currently works as intended, throughout its development it has faced several challenges - particularly related to the computational overhead and complexity that FHE brought. The first iterations of the system encountered multiple issues, with significant impacts to the systems response time and scalability. However, through multiple iterations and testing, these challenges were addressed and solutions found, resulting in enhanced performance and efficiency in the final system.

The rigorous testing process, included multiple iterations of security and functionality tests which played a crucial role in finalising the system. Each testing phase helped build on the system and make each iteration a more polished and refined system. Providing valuable feedback that led to the final systems completion. The final interaction of testing confirmed the robustness of the system, with it having a high efficiency in handling secure biometric data while also providing a smooth user interaction.

This system has successfully laid a solid foundation for future iterations and improvements. The integration of FHE presents a number of opportunities for the further innovation of this project, especially in the development of a more user-friendly interface and expanding the system to work on mobile and other devices. Furthermore the continuous research into reducing latency and computational demands will allow for this project to only shine more in the coming years.

In conclusion, this project has represented a significant stride forward in the realm of privacy-preserving technology. By successfully integrating FHE into the world of biometric authentication, the system not only meets the world's current demands for data security but also will benefit from future discoveries and research. The project's focus on security, user privacy and efficiency highlights the tackling of the most common challenges when working with biometric data,

paving a new path for a more secure and private authentication method for the future of the technological world.

References

1. *Jain, A., Bolle, R. and Pankanti, S. (1996) Introduction to biometrics, SpringerLink. Available at: https://link.springer.com/chapter/10.1007/0-306-47044-6_1 (Accessed: 19 March 2024).*
2. *Boneh, D. et al. (1970) Private database queries using somewhat homomorphic encryption, SpringerLink. Available at: https://link.springer.com/chapter/10.1007/978-3-642-38980-1_7 (Accessed: 19 March 2024).*
3. *smh767 (no date) Digital Shred. Available at: https://sites.psu.edu/digitalshred/2020/11/13/privacy-by-design-pbd-the-7-foundational-principles-cavoukian/ (Accessed: 19 March 2024).*
4. *University, C.G.S. et al. (2009) Fully homomorphic encryption using ideal lattices: Proceedings of the forty-first annual ACM Symposium on Theory of Computing, ACM Conferences. Available at: https://dl.acm.org/doi/abs/10.1145/1536414.1536440 (Accessed: 19 March 2024).*
5. *Login (no date) AIS Electronic Library AISeL. Available at: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1215&context=icis2006 (Accessed: 19 March 2024).*
6. *Michael Naehrig Eindhoven University of Technology et al. (2011) Can homomorphic encryption be practical?: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, ACM Conferences. Available at: https://dl.acm.org/doi/abs/10.1145/2046660.2046682 (Accessed: 19 March 2024).*
7. *Author links open overlay panelAnil K. Jain 1 a et al. (2016) 50 years of biometric research: Accomplishments, challenges, and opportunities, Pattern Recognition Letters. Available at: https://www.sciencedirect.com/science/article/pii/S0167865515004365?casa_token=H9ezKO89Q_wAAAAA%3AXiqFuLXFaUYIBXmiJLtrYKLELE3hhomgUknglKzLJG8RUd9UoaWBUVCEBOZSqDH7QsKH194nil4 (Accessed: 20 March 2024).*
8. *Acquisti, A., Taylor, C. and Wagman, L. (no date) The Economics of Privacy, Journal of Economic Literature. Available at: https://www.aeaweb.org/articles?id=10.1257%2Fjel.54.2.442 (Accessed: 20 March 2024).*
9. *Maltoni, D. et al. (no date) Handbook of Fingerprint Recognition, SpringerLink. Available at: https://link.springer.com/book/10.1007/978-3-030-83624-5 (Accessed: 20 March 2024).*
10. *Python for Data Analysis, 3E (no date) Wes McKinney. Available at: https://wesmckinney.com/book/ (Accessed: 03 April 2024).*
11. *(No date a) Somewhat homomorphic encryption. Available at: https://courses.csail.mit.edu/6.857/2017/project/22.pdf (Accessed: 03 April 2024).*

12. *(No date a) Fully homomorphic encryption for mathematicians. Available at: https://eprint.iacr.org/2013/250.pdf (Accessed: 03 April 2024).*
13. *Mohammad@Habibzadeh.us (no date) Biometric system features, Biometric system features | Palizafzar Corp. Available at: https://www.palizafzar.com/en/articles/biometric-technology-features/ (Accessed: 11 April 2024).*
14. *Sprint (2022) Agile Product Management | Definition and Overview. Available at: https://www.productplan.com/glossary/sprint/ (Accessed: 15 April 2024).*
15. *(No date) [PDF] which ring based somewhat homomorphic encryption scheme is best? | semantic scholar. Available at: https://www.semanticscholar.org/paper/Which-Ring-Based-Somewhat-Homomorphic-Encryption-is-Costache-Smart/0e2be1b38467743228f933cfeaac90a2baa3fdf2 (Accessed: 30 April 2024).*
16. *Laboratories, K.D.B.R. et al. (2009) Hail: Proceedings of the 16th ACM Conference on Computer and Communications Security, ACM Conferences. Available at: https://dl.acm.org/doi/abs/10.1145/1653662.1653686?casa_token=boZnLbAPsgAAAAAA%3AvJ6zPaSMArk1GOBPczZ83JL2k_v0jUjKQD1AQEHGCe_eIGmBr32df8fodbvdPY8j5585cOfCqrUK_A (Accessed: 30 April 2024).*

List of Figures

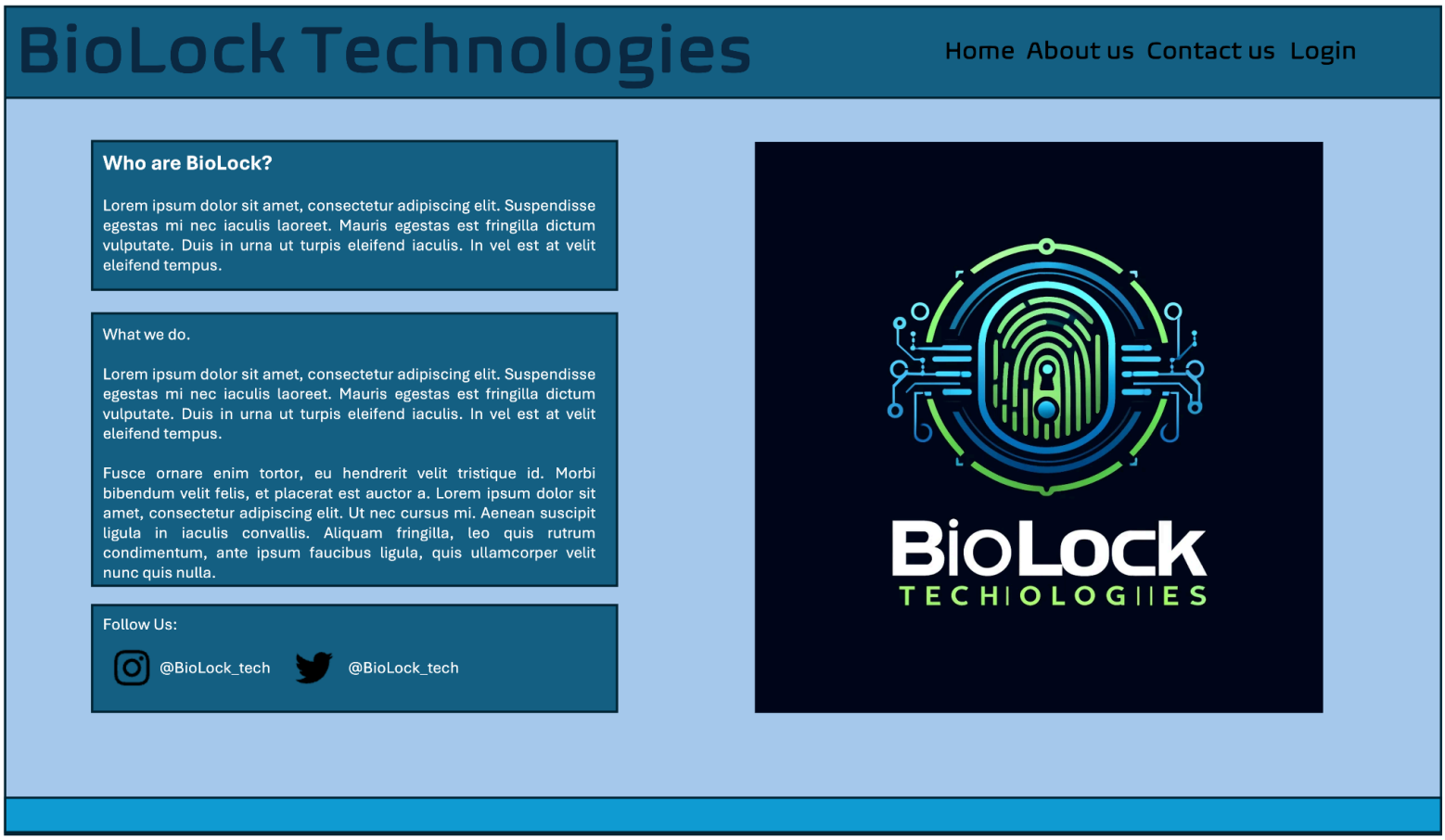
	9. Back-end flowchart	
	10. CLI Interaction	
Development Lifecycle + Iterations	11. Agile Methodology	*Sprint (2022) Agile Product Management | Definition and Overview. Available at: https://www.productplan.com/glossary/sprint/ (Accessed: 15 April 2024).*
	12. First Iteration Snippet	
	13. Second Iteration Snippet	
	14. Final Iteration Snippet	
Implementation	15. GUI structure	
	16. Sign In Function	
	17. Input Placement	
	18. Error Handling	
	19. CKKS Scheme	*Huynh, D. (2021) CKKS explained: Part 1, Vanilla Encoding and decoding, OpenMined Blog. Available at: https://blog.openmined.org/ckks-explained-part-*

		1-simple-encoding-and-decoding/ (Accessed: 03 April 2024).
	20. Create Keys	
	21. Serialisation	
	22. Encrypted Operations	

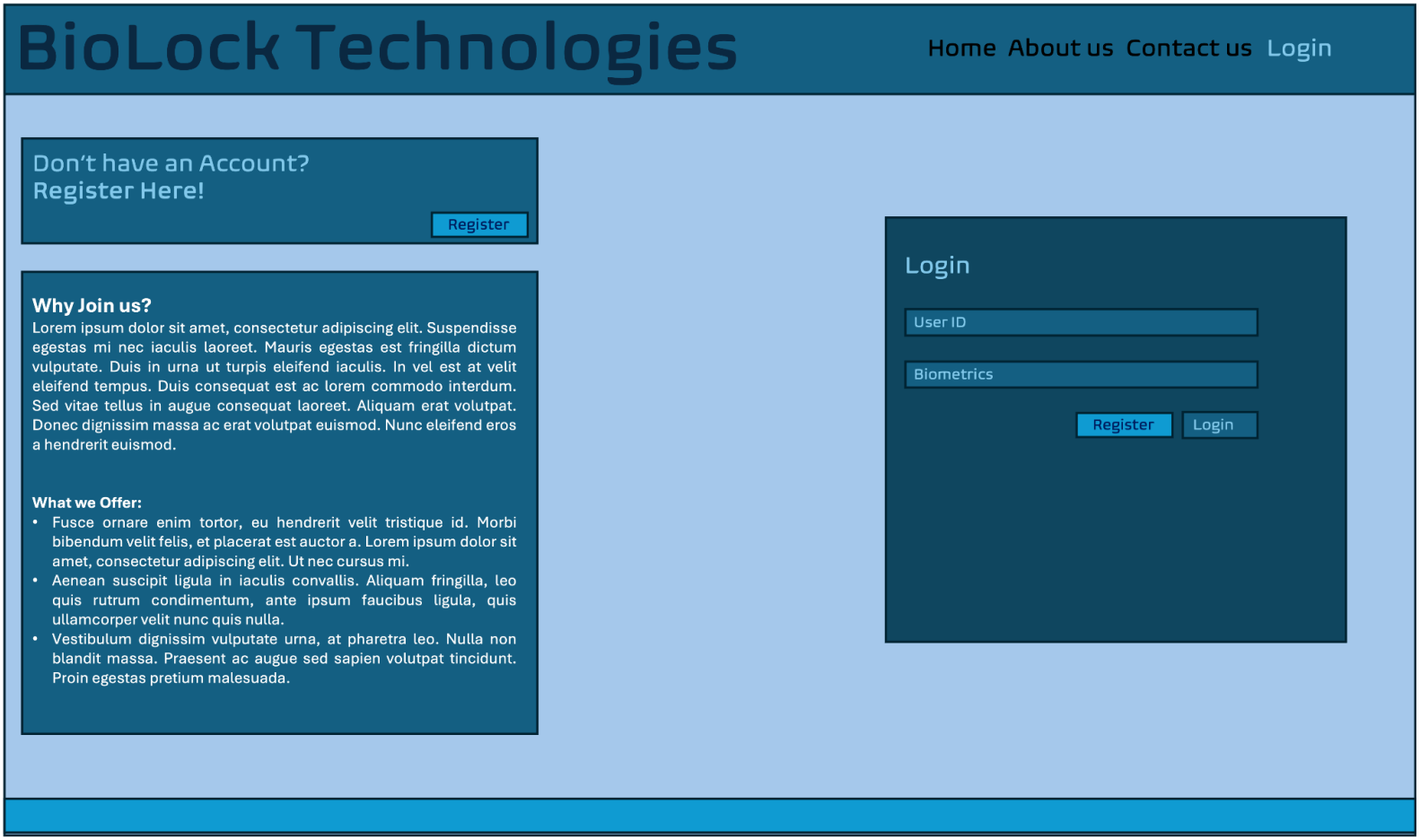
Test Cases: One Drive Folder

Appendices

1. Future Web Implementation

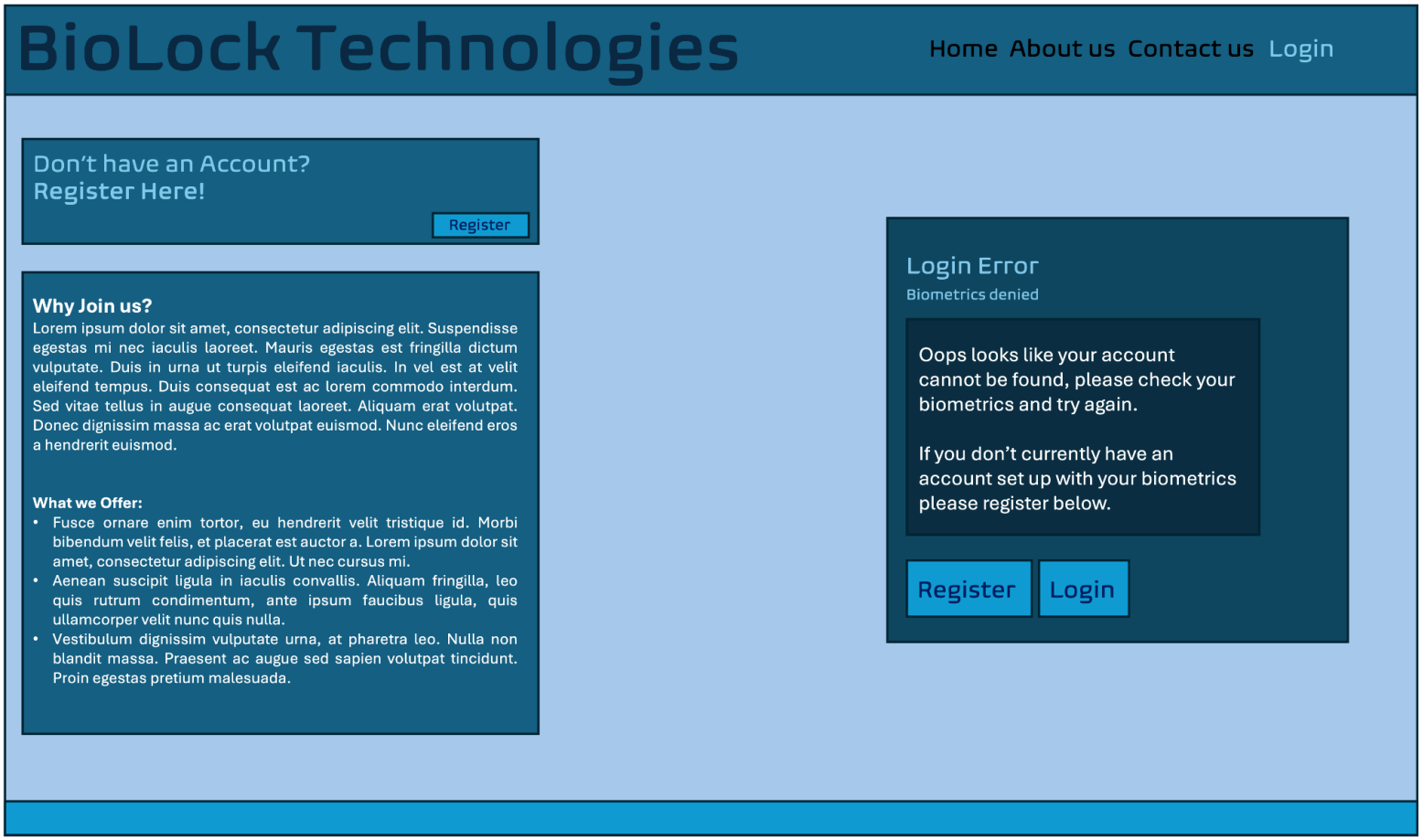


1.1 Login

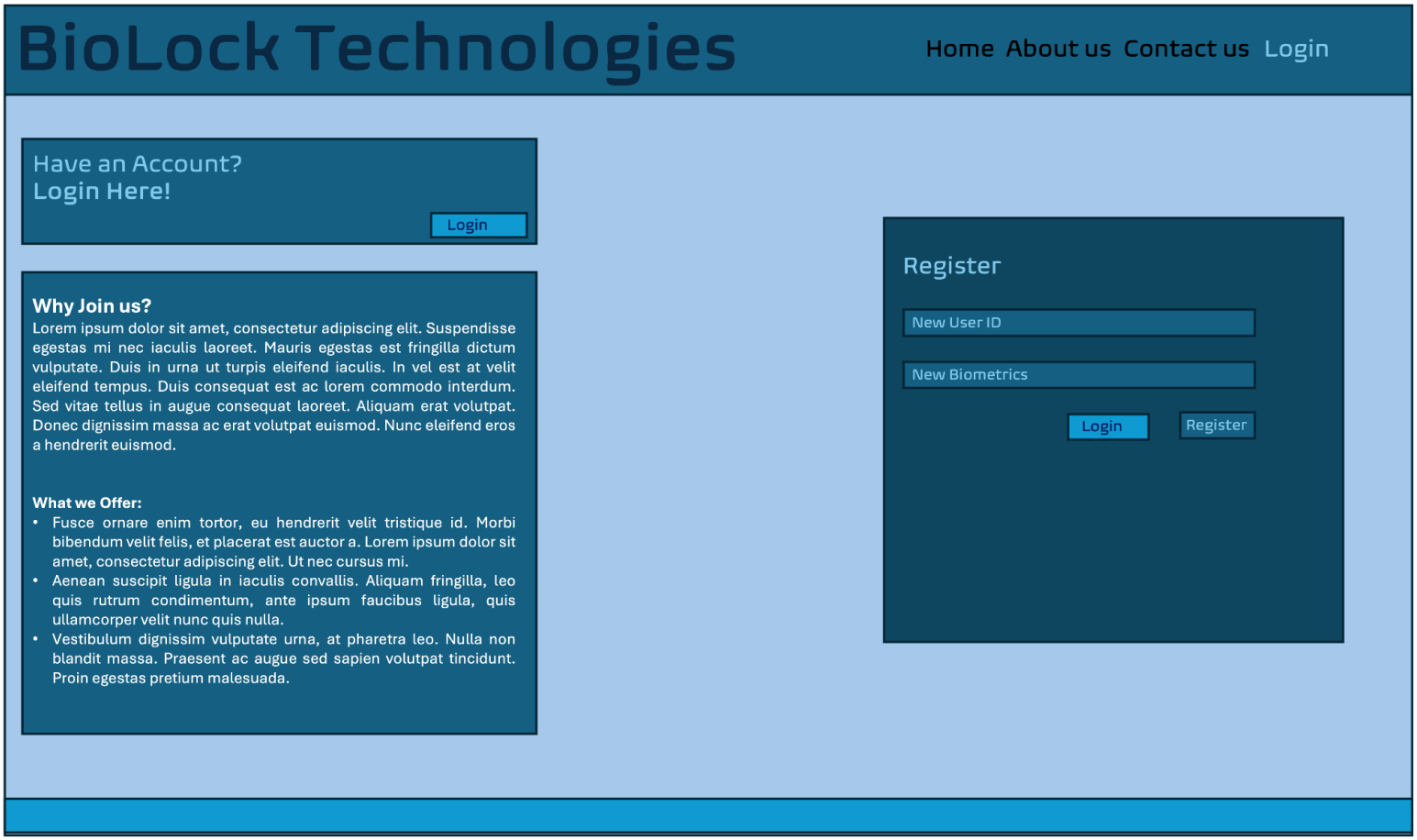

BioLock Technologies
Home About us Contact us Login
Don't have an Account?
Register Here!
Register
Why Join us?
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse egestas mi nec iaculis laoreet. Mauris egestas est fringilla dictum vulputate. Duis in urna ut turpis eleifend iaculis. In vel est at velit eleifend tempus. Duis consequat est ac lorem commodo interdum. Sed vitae tellus in augue consequat laoreet. Aliquam erat volutpat. Donec dignissim massa ac erat volutpat euismod. Nunc eleifend eros a hendrerit euismod.
What we Offer:
• Fusce ornare enim tortor, eu hendrerit velit tristique id. Morbi bibendum velit felis, et placerat est auctor a. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut nec cursus mi.
• Aenean suscipit ligula in iaculis convallis. Aliquam fringilla, leo quis rutrum condimentum, ante ipsum faucibus ligula, quis ullamcorper velit nunc quis nulla.
• Vestibulum dignissim vulputate urna, at pharetra leo. Nulla non blandit massa. Praesent ac augue sed sapien volutpat tincidunt. Proin egestas pretium malesuada.
Login
User ID
Biometrics
Register
Login

1.2 Login Successful Website UI

1.3 Login Fail Website UI



1.4 Register Website UI



2. First Iteration

```python
import random
import tenseal as ts

# Functions for file operations.
# These are utility functions that abstract the file I/O operations,
# allowing us to read and write binary data from and to files.
# This is particularly useful for storing encrypted data which is typically
# in a binary format.
def write_data(file_name, data):
    with open(file_name, 'wb') as file:
        file.write(data)

def read_data(file_name):
    with open(file_name, 'rb') as file:
        return file.read()

# Function to simulate fingerprint data capture.
# In a production environment, this function would interface with a biometric sensor to capture
# a fingerprint image or its features. Here, it generates a list of 10 random floating-point numbers
# between 0 and 1 to simulate a fingerprint's unique data points.
def capture_mock_fingerprint_data():
    return [random.uniform(0, 1) for _ in range(10)]

# Function to preprocess fingerprint data.
# This placeholder function represents the step where raw biometric data would
# undergo preprocessing such as normalization, feature extraction, or any other form of
# transformation to make it suitable for encryption and comparison. The current implementation
# simply returns the data as-is, but it's here to illustrate where such preprocessing logic would be inserted.
def preprocess_fingerprint_data(fingerprint_data):
    return fingerprint_data

# Function to create a TenSEAL context.
# This function initializes a TenSEAL context with specific parameters for the CKKS scheme,
# which allows computations on encrypted real numbers. It sets up the scheme, generates the necessary keys,
# and establishes a global scale which is used during encryption and decryption processes.
# The context generated here is a critical component that encapsulates the encryption environment.
def create_context():
    context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree=16384, coeff_mod_bit_sizes=[60, 40, 40, 60])
    context.generate_galois_keys()
    context.global_scale = 2**40
    return context
```

2.1 First Iteration

```python
# Function to encrypt biometric data.
# This function takes the biometric data, which should be a list of floating-point numbers,
# and a TenSEAL context. It uses the context to create a CKKS vector, which is the encrypted
# representation of the biometric data, and then serializes it into a byte stream that can be
# stored or transmitted securely.
def encrypt_biometric_data(data, context):
    encrypted_data = ts.ckks_vector(context, data)
    return encrypted_data.serialize()

# Function to decrypt data.
# This function accepts a serialized CKKS vector and the TenSEAL context used to encrypt it.
# It deserializes the vector, associates it with the context (linking it to the correct encryption parameters
# and keys), and then decrypts it to recover the original floating-point values. The decrypted data is then returned
# for further processing or comparison.
def decrypt_result(encrypted_result, context):
    auth_result = ts.lazy_ckks_vector_from(encrypted_result)
    auth_result.link_context(context)
    decrypted_result = auth_result.decrypt()
    return decrypted_result

# Main function for the PPBA system.
# This is the primary function that coordinates the various steps of the system: starting the process,
# inputting data, encrypting the data, performing secure computations, and optionally decrypting and validating the data.
# It is designed to be user-interactive, with input prompts guiding the user through the process.
# The function demonstrates the potential flow of a biometric authentication system that utilizes FHE for data privacy.
def privacy_preserving_biometric_authentication():
    start = input("Start? (yes/no): ")
    if start.lower() != "yes":
        print("Exited.")
        return

    input_data = input("Do you want to input fingerprint data? (yes/no): ")
    if input_data.lower() == "yes":
        fingerprint_data = list(map(float, input("Input Fingerprint Data (numerical values separated by space): ").split()))
        fingerprint_data = preprocess_fingerprint_data(fingerprint_data)

        encrypt = input("Do you want to encrypt? (yes/no): ")
        if encrypt.lower() == "yes":
            context = create_context()
            encrypted_data = encrypt_biometric_data(fingerprint_data, context)

            user_id = input("Enter user ID for storing the data: ")
            write_data(f'{user_id}_encrypted_fingerprint.txt', encrypted_data)
```

2.2 First Iteration

```
            # Secure computations are demonstrated here. We perform an addition and a multiplication
            # on the encrypted data. These operations are chosen to show how FHE allows us to compute
            # on ciphertexts. The add_value and mul_value are random numbers that simulate some transformation
            # of the data while it remains encrypted.
            add_value = random.randint(1, 20)
            mul_value = random.randint(1, 5)
            encrypted_vector = ts.lazy_ckks_vector_from(encrypted_data)
            encrypted_vector.link_context(context)
            encrypted_vector.add_(add_value)
            encrypted_vector.mul_(mul_value)
            print(f"Performed addition operation on the encrypted data with value: {add_value}")
            print(f"Performed multiplication operation on the encrypted data with value: {mul_value}")

            decrypt = input("Do you want to decrypt? (yes/no): ")
            if decrypt.lower() == "yes":
                decrypted_data_with_operations = decrypt_result(encrypted_vector.serialize(), context)
                decrypted_data = [round((x / mul_value) - add_value, 5) for x in decrypted_data_with_operations]

                print("Original Fingerprint Data:", fingerprint_data)
                print("Decrypted Data (after reversing the operations):", decrypted_data)

                if all(round(original, 5) == decrypted for original, decrypted in zip(fingerprint_data, decrypted_data)):
                    print("Successful: The decrypted output matches the original input.")
                else:
                    print("Unsuccessful: The decrypted output does not match the original input.")
            else:
                print("Exited without decrypting.")
        else:
            print("Exited without encrypting.")
    else:
        print("Exited without inputting data.")


privacy_preserving_biometric_authentication()
```

3. Second Iteration

```
import json      You, 6 days ago • Trying to get the code to work when matching st…
import tenseal as ts
import base64
import zlib
import logging

DATABASE_FILE_PATH = '/Users/lukedawson1156/Desktop/Yr 3 Project/Development Project/fingerprint_database.txt'

logging.basicConfig(level=logging.DEBUG, filename='biometric_system.log', filemode='w',
                    format='%(asctime)s - %(levelname)s - %(message)s')

def save_database(database, file_path):
    logging.debug("Saving database to %s", file_path)
    # Convert the bytes data to base64 encoded strings and compress before saving
    encoded_and_compressed_database = {
        k: base64.b64encode(zlib.compress(v, level=9)).decode('utf-8') for k, v in database.items()
    }
    with open(file_path, 'w') as file:
        json.dump(encoded_and_compressed_database, file)

def load_database(file_path):
    logging.debug("Loading database from %s", file_path)
    try:
        with open(file_path, 'r') as file:
            encoded_and_compressed_database = json.load(file)
        # Decompress and decode the base64 encoded strings back to bytes
        return {
            k: zlib.decompress(base64.b64decode(v.encode('utf-8')))
            for k, v in encoded_and_compressed_database.items()
        }
    except (FileNotFoundError, json.JSONDecodeError):
        return {}  # Returns an empty dictionary if the file doesn't exist or if the JSON is invalid
    except zlib.error:
        print("Decompression error. Database may be corrupted.")
        return {}

fingerprint_database = load_database(DATABASE_FILE_PATH)
```

3.1 Second Iteration

```
def create_context_and_keys():
    #Initialises a TenSEAL context with CKKS scheme and generates necessary keys.
    context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree=8192, coeff_mod_bit_sizes=[60, 40, 40, 60])
    context.generate_galois_keys()
    context.global_scale = 2**40
    return context


def preprocess_biometric_data(data):
    logging.debug("Preprocessing biometric data: %s", data)
    # Example: Assuming the biometric data ranges from 0 to 100.
    # Adjust these min and max values based on your actual data range.
    min_val = 0
    max_val = 100
    normalized_data = [(float(i) - min_val) / (max_val - min_val) for i in data]
    return normalized_data

def encrypt_biometric_data(data, context):
    logging.debug("Encrypting biometric data")
    encrypted_data = ts.ckks_vector(context, data)
    return encrypted_data.serialize()

def decrypt_data(encrypted_data, context):
    logging.debug("Decrypting data")
    #Decrypts the given encrypted data using the provided TenSEAL context.
    encrypted_vector = ts.lazy_ckks_vector_from(encrypted_data)
    encrypted_vector.link_context(context)
    decrypted_data = encrypted_vector.decrypt()
    return [round(num, 2) for num in decrypted_data]

def perform_encrypted_operations(encrypted_data, context):
    #Performs predefined homomorphic operations on the encrypted data.
    encrypted_vector = ts.lazy_ckks_vector_from(encrypted_data)
    encrypted_vector.link_context(context)
    encrypted_vector.add_(10)
    encrypted_vector.mul_(2)
    return encrypted_vector.serialize()
```

3.2 Second Iteration

```
def input_biometric_data():
    #Simulates biometric data capture by allowing the user to input a string of 5 integers.
    #Preprocesses the input data for normalization.
    print("Please enter your biometric data as 5 numbers separated by space (each number represents a biometric point):")
    data = list(map(float, input().split()))
    if len(data) != 5:
        print("Invalid input. Please enter exactly 5 numbers.")
        return input_biometric_data()
    preprocessed_data = preprocess_biometric_data(data)
    return preprocessed_data

def print_all_user_ids():
    # Prints all User IDs that are currently stored in the database.
    if fingerprint_database:
        print("User IDs currently stored in the database:")
        for user_id in fingerprint_database.keys():
            print(user_id)
    else:
        print("No fingerprints are currently stored in the database.")

def compare_biometric_data(stored_data, input_data, threshold=0.95):
    logging.debug("Comparing biometric data: stored_data=%s, input_data=%s, threshold=%s", stored_data, input_data, threshold)
    # Compares stored biometric data with the input data to determine if they match with 95%+ correctness.
    differences = [abs(a - b) for a, b in zip(stored_data, input_data)]
    avg_difference = sum(differences) / len(differences)
    match_threshold = 1 - threshold
    return avg_difference <= match_threshold
```

3.3 Second Iteration

```
def user_interaction_flow(context):
    while True:
        action = input("Choose an action: \n1. Add new ID \n2. Log in with existing ID \n3. Exit \nYour choice (1/2/3): ").strip()

        if action == "1":
            # Logic to add a new ID
            user_id = input("Enter a User ID for your new fingerprint: ").strip()
            fingerprint_data = input_biometric_data()
            encrypted_data = encrypt_biometric_data(fingerprint_data, context)
            fingerprint_database[user_id] = encrypted_data
            save_database(fingerprint_database, DATABASE_FILE_PATH)
            print("New fingerprint stored successfully. Your data has been encrypted.")
            print_all_user_ids()

        elif action == "2":
            # Logic to "log in" or verify an existing ID
            user_id = input("Please enter your User ID: ").strip()
            if user_id in fingerprint_database:
                print("Please enter your biometric data for verification:")
                fingerprint_data = input_biometric_data()
                encrypted_data = encrypt_biometric_data(fingerprint_data, context)
                stored_encrypted_data = fingerprint_database[user_id]
                stored_fingerprint_data = decrypt_data(stored_encrypted_data, context)
                current_fingerprint_data = decrypt_data(encrypted_data, context)
                if compare_biometric_data(stored_fingerprint_data, current_fingerprint_data):
                    print("Biometrics approved.")
                else:
                    print("Error: Biometrics denied.")
            else:
                print("No user ID found in the database.")

        elif action == "3":
            # Exiting the system
            print("Exiting the system.")
            break
        else:
            print("Invalid response. Please enter '1', '2', or '3'.")
```

3.4 Second Iteration

```
def store_or_verify_fingerprint(context):
    global fingerprint_database
    logging.info("User action: %s", action)
    if action == "no":
        user_id = input("Enter a User ID for your new fingerprint: ").strip()
        fingerprint_data = input_biometric_data()
        encrypted_data = encrypt_biometric_data(fingerprint_data, context)
        fingerprint_database[user_id] = encrypted_data
        save_database(fingerprint_database, DATABASE_FILE_PATH)
        print("New fingerprint stored successfully.")
        print_all_user_ids()
        print("Your data has been encrypted.")
    elif action == "yes":
        user_id = input("Please enter your User ID: ").strip()
        if user_id in fingerprint_database:
            print("Please enter your biometric data for verification:")
            fingerprint_data = input_biometric_data()
            encrypted_data = encrypt_biometric_data(fingerprint_data, context)
            stored_encrypted_data = fingerprint_database[user_id]
            stored_fingerprint_data = decrypt_data(stored_encrypted_data, context)
            current_fingerprint_data = decrypt_data(encrypted_data, context)
            if compare_biometric_data(stored_fingerprint_data, current_fingerprint_data):
                print("Biometrics approved.")
            else:
                print("Error: Biometrics denied.")
        else:
            print("No user ID found in the database.")

# Main Function Demonstrating PPBA System
def privacy_preserving_biometric_authentication():
    context = create_context_and_keys()
    user_interaction_flow(context)

if __name__ == "__main__":
    privacy_preserving_biometric_authentication()
```

4. Third Iteration

```
import tenseal as ts
import sqlite3
import zlib
import base64
import logging
import os

# Paths for database, keys, and log file
DATABASE_PATH = '/Users/lukedawson1156/Desktop/Yr 3 Project/Development Project/Iterations/Database/fingerprints-Main.db'
KEYS_FILE_PATH = '/Users/lukedawson1156/Desktop/Yr 3 Project/Development Project/Iterations/Database/Keys/print-key-Main.bin'
LOG_FILE_PATH = '/Users/lukedawson1156/Desktop/Yr 3 Project/Development Project/Iterations/Logs/biometric_system-Main.log'

# Ensure the log file directory exists
os.makedirs(os.path.dirname(LOG_FILE_PATH), exist_ok=True)

# Setup logging for debugging and tracking
logging.basicConfig(
    level=logging.DEBUG,
    filename=LOG_FILE_PATH,
    filemode='a',
    format='%(asctime)s - %(levelname)s - %(message)s',
)

def initialize_database():
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS fingerprints (
                        user_id TEXT PRIMARY KEY,
                        encrypted_data BLOB
                      )''')
    conn.commit()
    conn.close()
    logging.debug("Database initialized and table created if not already present.")

def save_to_database(user_id, encrypted_data):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    compressed_data = zlib.compress(encrypted_data)
    encoded_data = base64.b64encode(compressed_data)
    blob_data = sqlite3.Binary(encoded_data)
    cursor.execute("REPLACE INTO fingerprints (user_id, encrypted_data) VALUES (?, ?)",
                   (user_id, blob_data))
    conn.commit()
    conn.close()
    logging.debug(f"Data saved for user_id {user_id}")
```

4.1 Third Iteration

```
def load_from_database(user_id):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT encrypted_data FROM fingerprints WHERE user_id = ?", (user_id,))
    row = cursor.fetchone()
    conn.close()
    if row and isinstance(row[0], bytes):
        decompressed_data = zlib.decompress(base64.b64decode(row[0]))
        logging.debug(f"Data loaded for user_id {user_id}")
        return decompressed_data
    logging.debug(f"No data found for user_id {user_id}")
    return None

ions (⇧⌘X)  _keys(context):
    with open(KEYS_FILE_PATH, 'wb') as key_file:
        key_file.write(context.serialize(save_secret_key=True))
    logging.debug("Encryption keys saved.")

def load_keys():
    with open(KEYS_FILE_PATH, 'rb') as key_file:
        logging.debug("Encryption keys loaded.")
        return ts.context_from(key_file.read())

def create_context_and_keys():
    if os.path.exists(KEYS_FILE_PATH):
        return load_keys()
    else:
        context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree=8192, coeff_mod_bit_sizes=[60, 40, 40, 60])
        context.generate_galois_keys()
        context.global_scale = 2**40
        save_keys(context)
        logging.debug("New encryption context and keys created.")
        return context

def perform_encrypted_comparison(encrypted_data1, encrypted_data2, context):
    vector1 = ts.ckks_vector_from(context, encrypted_data1)
    vector2 = ts.ckks_vector_from(context, encrypted_data2)

    difference_vector = vector1 - vector2
    sum_vector = vector1 + vector2
    product_vector = vector1 * vector2

    decrypted_differences = difference_vector.decrypt()
    decrypted_sums = sum_vector.decrypt()
    decrypted_products = product_vector.decrypt()

    logging.debug(f"Decrypted differences: {decrypted_differences}")
    logging.debug(f"Decrypted sums: {decrypted_sums}")
    logging.debug(f"Decrypted products: {decrypted_products}")

    return decrypted_differences, decrypted_sums, decrypted_products
```

4.2 Third Iteration

```
def user_interaction_flow(context):
    while True:
        action = input("Choose an action:\n1. Add user\n2. Log in\n3. Exit\nYour choice (1/2/3): ").strip()

        if action == "1":
            user_id = input("Enter a new User ID: ").strip()
            biometric_data = [float(x) for x in input("Enter your 3 biometric digits as space-separated numbers: ").strip().split()]
            logging.debug(f"Biometric data input for user_id {user_id}: {biometric_data}")

            encrypted_data = ts.ckks_vector(context, biometric_data).serialize()
            logging.debug(f"Encrypted data for user_id {user_id}: {encrypted_data}")

            save_to_database(user_id, encrypted_data)
            logging.debug(f"New user {user_id} registered and data encrypted and stored.")

        elif action == "2":
            user_id = input("Enter your User ID: ").strip()
            stored_encrypted_data = load_from_database(user_id)
            if stored_encrypted_data:
                biometric_data = [float(x) for x in input("Enter your 3 biometric digits for verification as space-separated numbers: ").strip().split()]
                current_encrypted_data = ts.ckks_vector(context, biometric_data).serialize()
                differences, _, _ = perform_encrypted_comparison(current_encrypted_data, stored_encrypted_data, context)
                if all(abs(diff) < 0.1 for diff in differences):
                    print("Biometric authentication successful.")
                else:
                    print("Biometric authentication failed.")
            else:
                print("No data found for this user ID.")

        elif action == "3":
            print("Exiting system.")
            break

        else:
            print("Invalid option selected. Please try again.")

def main():
    initialize_database()
    context = create_context_and_keys()
    user_interaction_flow(context)

if __name__ == "__main__":
    main()
```

5. Final Iteration

```
import logging
import tenseal as ts
import sqlite3
import zlib
import base64
import os
from tkinter import *
from tkinter import messagebox


# Configure logging
logging.basicConfig(filename='app.log', level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

# Database and encryption key paths
DATABASE_PATH = 'C:\\Database\\prints-MainW.db'
KEYS_FILE_PATH = 'C:\\Database\\print-key-MainW.bin'

def initialize_database():
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS fingerprints (
                        user_id TEXT PRIMARY KEY,
                        encrypted_data BLOB
                    )''')
    conn.commit()
    conn.close()
    logging.info("Database initialized.")

def save_to_database(user_id, encrypted_data):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    compressed_data = zlib.compress(encrypted_data)
    encoded_data = base64.b64encode(compressed_data)
    blob_data = sqlite3.Binary(encoded_data)
    cursor.execute("REPLACE INTO fingerprints (user_id, encrypted_data) VALUES (?, ?)", (user_id, blob_data))
    conn.commit()
    conn.close()
    logging.info(f"Data saved for user {user_id}")

def load_from_database(user_id):
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT encrypted_data FROM fingerprints WHERE user_id = ?", (user_id,))
    row = cursor.fetchone()
    conn.close()
    if row:
        decompressed_data = zlib.decompress(base64.b64decode(row[0]))
        logging.info(f"Data loaded for user {user_id}")
        return decompressed_data
    logging.warning(f"No data found for user {user_id}")
    return None
```

5.1 Final Iteration

```
def perform_encrypted_comparison(encrypted_data1, encrypted_data2, context):
    vector1 = ts.ckks_vector_from(context, encrypted_data1)
    vector2 = ts.ckks_vector_from(context, encrypted_data2)

    # Check if the sizes of the vectors match using the size() method
    if vector1.size() != vector2.size():
        logging.warning("Attempted to compare vectors of different sizes.")
        ret                                    utomatically fail authentication

    difference_vector = vector1 - vector2
    sum_vector = vector1 + vector2
    product_vector = vector1 * vector2

    decrypted_differences = difference_vector.decrypt()
    decrypted_sums = sum_vector.decrypt()
    decrypted_products = product_vector.decrypt()

    logging.debug(f"Decrypted differences: {decrypted_differences}")
    logging.debug(f"Decrypted sums: {decrypted_sums}")
    logging.debug(f"Decrypted products: {decrypted_products}")

    return all(abs(diff) < 0.1 for diff in decrypted_differences), decrypted_sums, decrypted_products

def create_context_and_keys():
    if os.path.exists(KEYS_FILE_PATH):
        with open(KEYS_FILE_PATH, 'rb') as key_file:
            context = ts.context_from(key_file.read())
            logging.info("Encryption context loaded from existing keys.")
            return context
    else:
        context = ts.context(ts.SCHEME_TYPE.CKKS, poly_modulus_degree=8192, coeff_mod_bit_sizes=[60, 40, 40, 60])
        context.generate_galois_keys()
        context.global_scale = 2**40
        with open(KEYS_FILE_PATH, 'wb') as key_file:
            key_file.write(context.serialize(save_secret_key=True))
        logging.info("New encryption context and keys generated.")
        return context

context = create_context_and_keys()
initialize_database()

root = Tk()
root.title('Biometric Login')
root.geometry('925x500+300+200')
root.configure(bg='#003366')
root.resizable(False, False)
```

Click to collapse the range.

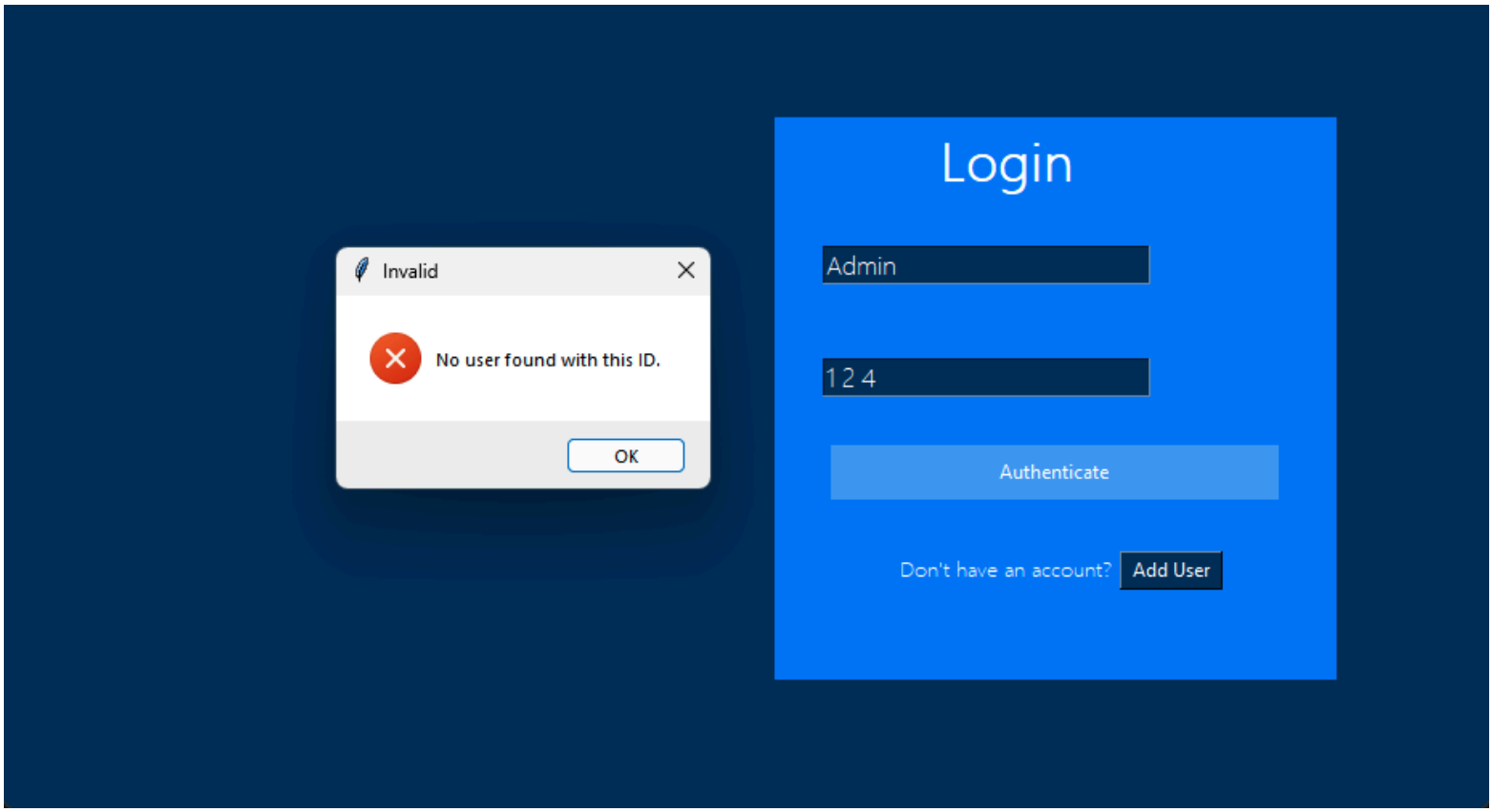
5.2 Final Iteration

```
def signin():
    username = user.get()
    data = [float(x) for x in code.get().split()]
    encrypted_input_data = ts.ckks_vector(context, data).serialize()
    encrypted_stored_data = load_from_database(username)
    if encrypted_stored_data:
        comparison_result, sums, products = perform_encrypted_comparison(encrypted_input_data, encrypted_stored_data, context)
        if comparison_result:
            messagebox.showinfo('Login Success', 'Biometric authentication successful.')
        else:
            messagebox.showerror('Invalid', 'Biometric authentication failed.')
            logging.error(f"Verification failed for user {username}")
    else:
        messagebox.showerror('Invalid', 'No user found with this ID.')

def add():
    window = Toplevel(root)
    window.title("Add User")
    window.geometry('925x500+300+200')
    window.configure(bg='#003366')
    window.resizable(False, False)

    def signup():
        username = user_add.get()
        data = code_add.get()
        data_list = [float(x) for x in data.split()]
        encrypted_data = ts.ckks_vector(context, data_list).serialize()
        save_to_database(username, encrypted_data)
        messagebox.showinfo('Signup', 'Successfully added user.')
        window.destroy()



    frame = Frame(window, width=350, height=370, bg='#007FFF')
    frame.place(x=480, y=70)

    heading = Label(frame, text='Add User', fg='#FFFFFF', bg='#007FFF', font=('Microsoft YaHei UI Light', 23, 'bold'))
    heading.place(x=100, y=5)

    user_add = Entry(frame, width=25, fg='#FFFFFF', border=1, bg='#003366', font=('Microsoft YaHei UI Light', 11))
    user_add.place(x=30, y=80)
    user_add.insert(0, 'User ID')
    user_add.bind('<FocusIn>', lambda e: user_add.delete(0, END) if user_add.get() == 'User ID' else None)
    user_add.bind('<FocusOut>', lambda e: user_add.insert(0, 'User ID') if not user_add.get() else None)
```

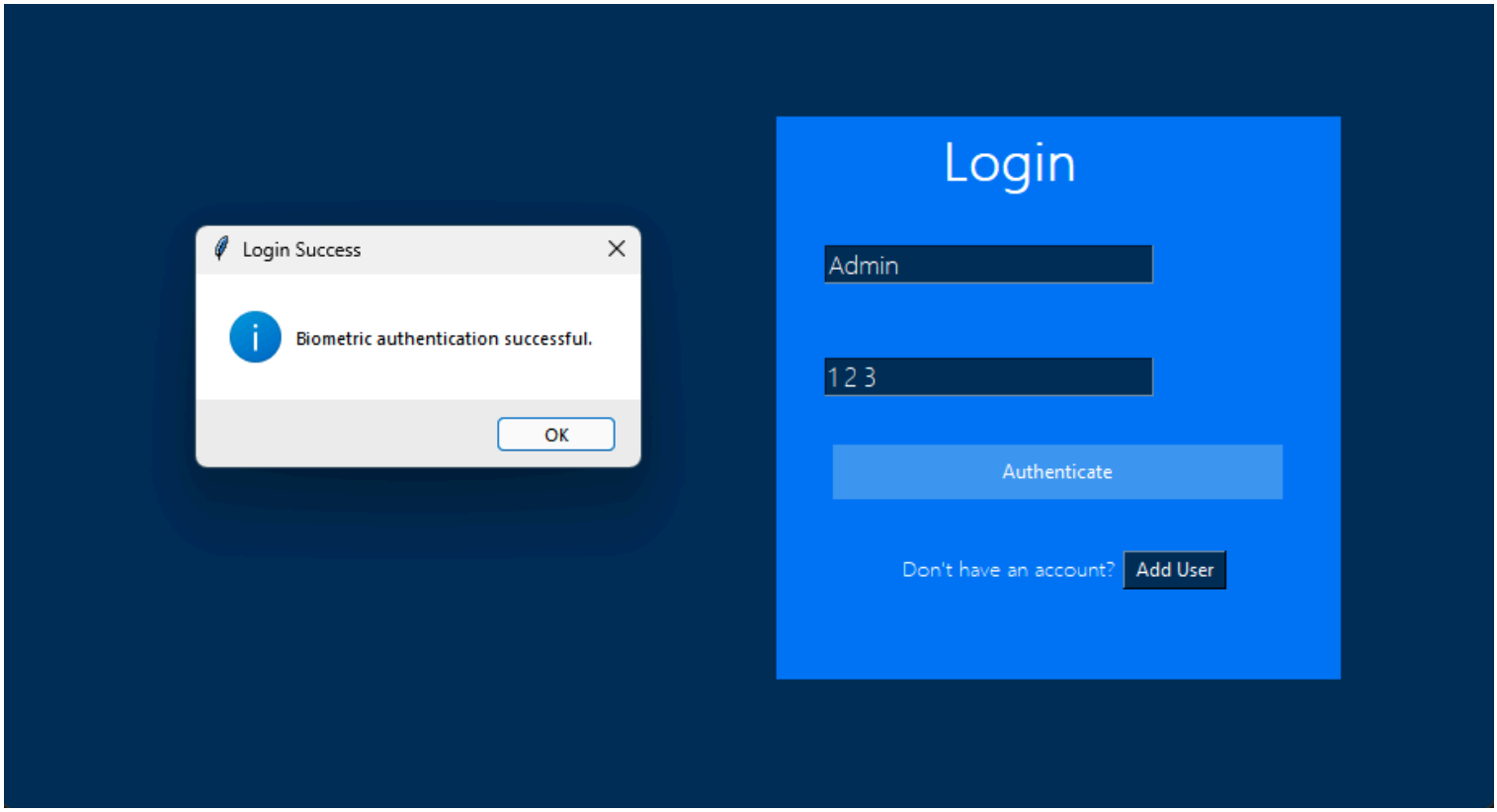
5.3 Final Iteration

```
    code_add = Entry(frame, width=25, fg='#FFFFFF', border=1, bg='#003366', font=('Microsoft YaHei UI Light', 11))
    code_add.place(x=30, y=150)
    code_add.insert(0, 'Biometric Data')
    code_add.bind('<FocusIn>', lambda e: code_add.delete(0, END) if code_add.get() == 'Biometric Data' else None)
    code_add.bind('<FocusOut>', lambda e: code_add.insert(0, 'Biometric Data') if not code_add.get() else None)

    Button(frame, width=39, pady=7, text='Add User', bg='#57a1f8', fg='white', border=0, command=signup).place(x=35, y=280)

    window.mainloop()

frame = Frame(root, width=350, height=350, bg='#007FFF')
frame.place(x=480, y=70)

heading = Label(frame, text='Login', fg='#FFFFFF', bg='#007FFF', font=('Microsoft YaHei UI Light', 23, 'bold'))
heading.place(x=100, y=5)

user = Entry(frame, width=25, fg='#FFFFFF', border=1, bg='#003366', font=('Microsoft YaHei UI Light', 11))
user.place(x=30, y=80)
user.insert(0, 'User ID')
user.bind('<FocusIn>', lambda e: user.delete(0, END) if user.get() == 'User ID' else None)
user.bind('<FocusOut>', lambda e: user.insert(0, 'User ID') if not user.get() else None)

code = Entry(frame, width=25, fg='#FFFFFF', border=1, bg='#003366', font=('Microsoft YaHei UI Light', 11))
code.place(x=30, y=150)
code.insert(0, 'Biometric Data')
code.bind('<FocusIn>', lambda e: code.delete(0, END) if code.get() == 'Biometric Data' else None)
code.bind('<FocusOut>', lambda e: code.insert(0, 'Biometric Data') if not code.get() else None)

Button(frame, width=39, pady=7, text='Authenticate', bg='#57a1f8', fg='white', border=0, command=signin).place(x=35, y=204)
label=Label(frame,text="Don't have an account?", fg='#FFFFFF', bg='#007FFF', font=('Microsoft YaHei UI Light', 9))
label.place(x=75,y=270)
add_user = Button(frame, width=8, text='Add User', border=1, bg='#003366', cursor='hand2', fg='#FFFFFF', command=add)
add_user.place(x=215, y=270)

root.mainloop()
```

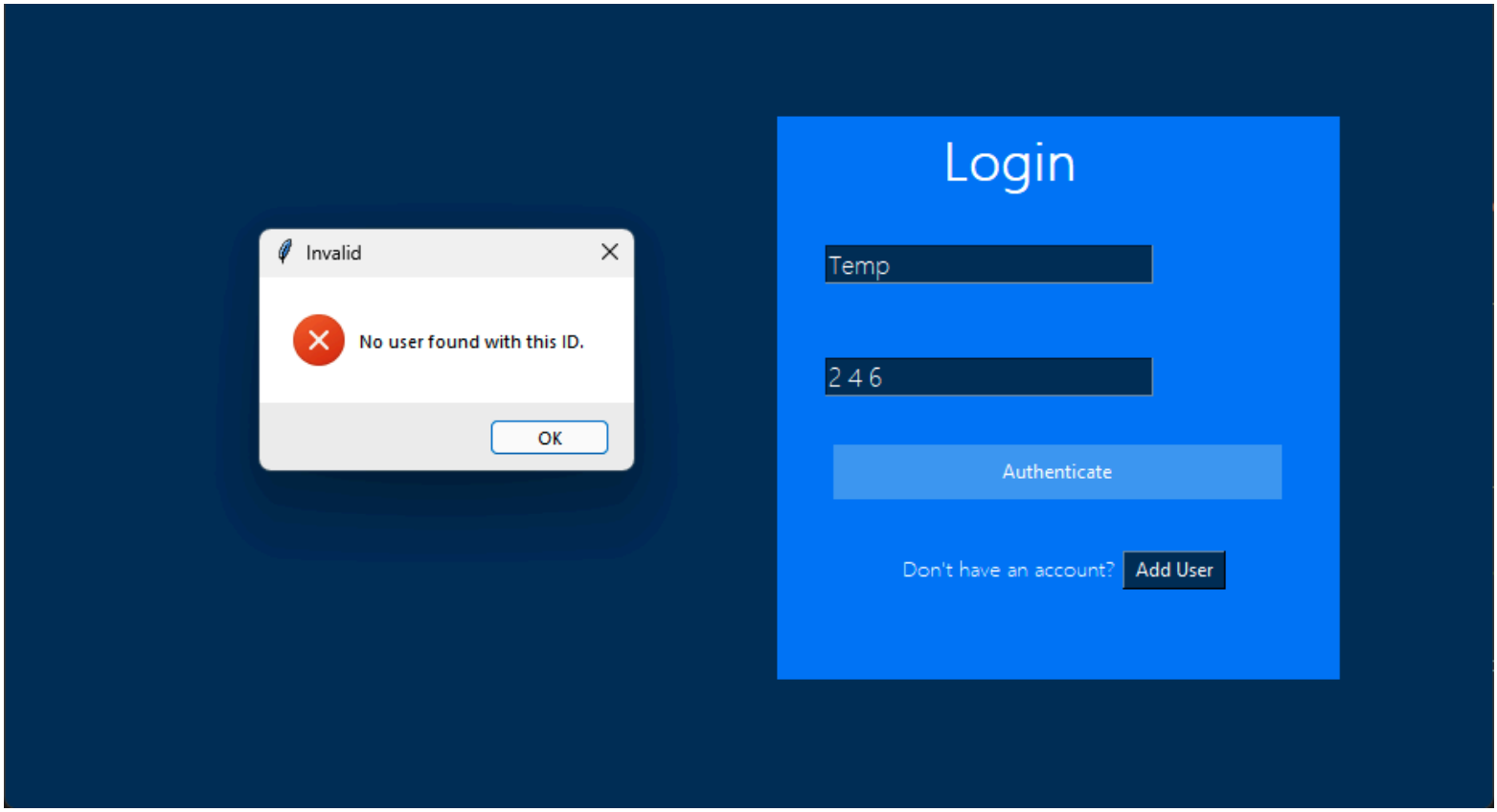
6. Use Case Tests



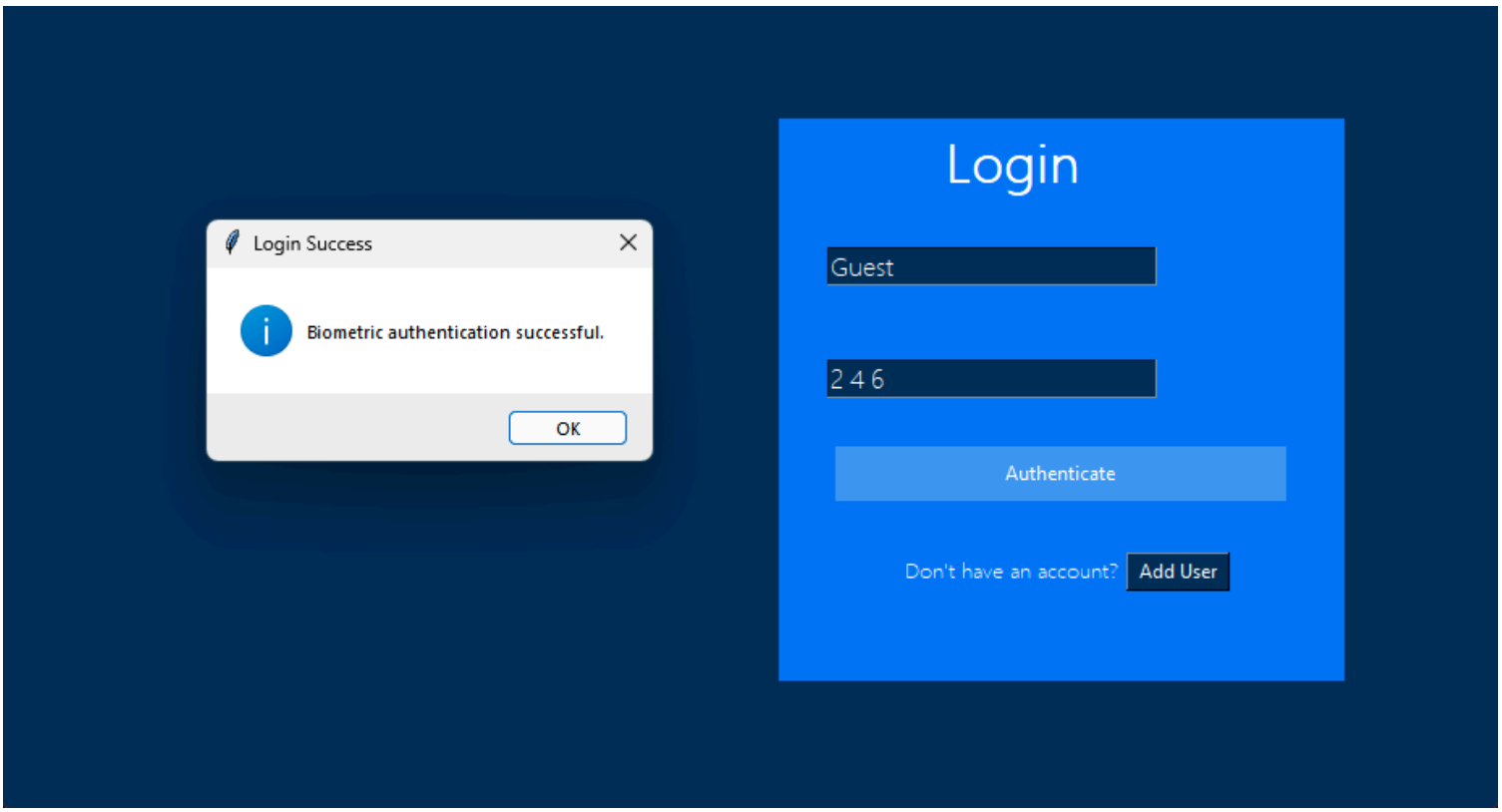
6.1 Use Case Test



6.2 Use Case Test



6.3 Use Case Test



6.4 Use Case Test