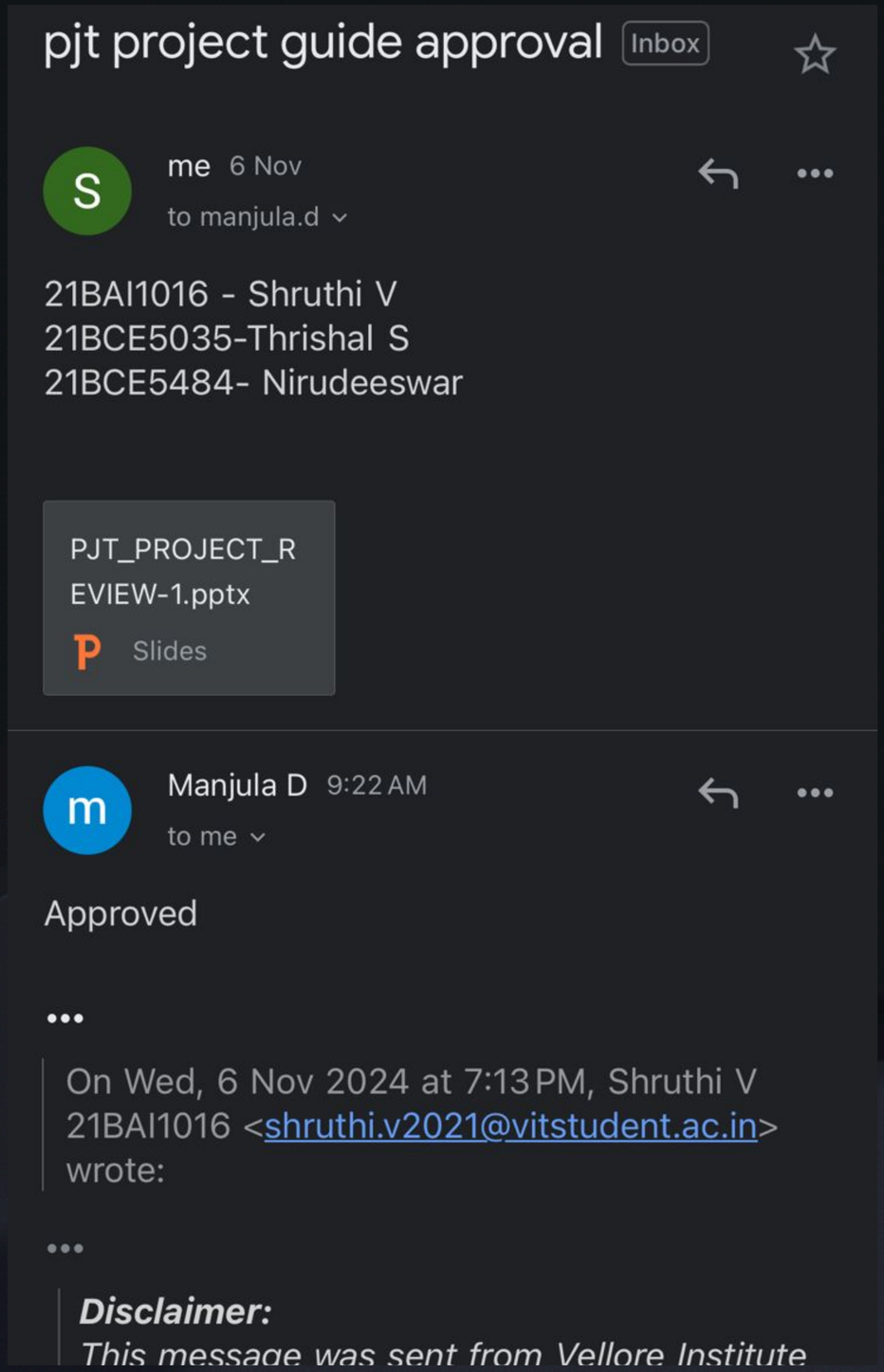
DETECTING LIVE FINGERPRINT SPOOFING IN REAL TIME

NIRUDEESWAR - 21BCE5484
THRISHAL - 21BCE5035
SHRUTHI - 21BAI1016

GUIDE : DR.MANJULA . D

PROJECT APPROVAL MAIL FROM GUIDE

- GUIDE - Dr.Manjula D
- Team:
  - NIRUDEESWAR - 21BCE5484
  - THRISHAL - 21BCE5035
  - SHRUTHI - 21BAI1016



pjt project guide approval Inbox

me 6 Nov
to manjula.d

21BAI1016 - Shruthi V
21BCE5035-Thrishal S
21BCE5484- Nirudeeswar

PJT_PROJECT_REVIEW-1.pptx
Slides

Manjula D 9:22 AM
to me

Approved

...

On Wed, 6 Nov 2024 at 7:13 PM, Shruthi V 21BAI1016 <shruthi.v2021@vitstudent.ac.in> wrote:

...

Disclaimer:
This message was sent from Vellore Institute

INTRODUCTION

- Fingerprint scanners, while popular, are susceptible to spoofing. We propose a multimodal biometric system integrating fingerprint recognition with DNA analysis from sweat, vein patterns, and physiological data (oxygen, heart rate) to combat spoofing and enhance security. This fusion significantly increases spoofing difficulty, strengthens authentication, and broadens application in high-security scenarios. While further development is needed, this approach offers a promising future for robust and versatile biometric authentication.

We propose a revolutionary biometric system that goes beyond fingerprints by integrating DNA patterns in sweat, vein patterns, and real-time physiological data like oxygen levels and heart rate.

This multi-layered approach significantly reduces spoofing risks and strengthens authentication.

With applications ranging from high-security facilities to personal devices, it promises a robust and versatile future for biometric authentication.

PROBLEM STATEMENT

1. TRADITIONAL FINGERPRINT SENSORS, WIDELY USED IN BIOMETRIC AUTHENTICATION SYSTEMS, ARE INCREASINGLY VULNERABLE TO SPOOFING ATTACKS, WHERE FAKE FINGERPRINTS CAN DECEIVE THESE SYSTEMS, LEADING TO UNAUTHORISED ACCESS.

2. THIS PROJECT AIMS TO DEVELOP A ROBUST, GUMMY FINGERS, REAL-TIME FINGERPRINT SENSING SYSTEM THAT LEVERAGES A COMBINATION OF RADIO FREQUENCY (RF) SIGNALS, RAMAN SPECTROSCOPY, AND DNA PATTERN ANALYSIS FROM SWEAT TO ACCURATELY DISTINGUISH BETWEEN LIVE AND SPOOFED FINGERPRINTS.

3. THE CHALLENGE LIES IN INTEGRATING THESE ADVANCED DETECTION METHODS INTO AN EMBEDDED SYSTEM CAPABLE OF PROCESSING COMPLEX BIOMETRIC DATA SWIFTLY AND SECURELY, WHILE ADDRESSING PRIVACY CONCERNS RELATED TO THE USE OF GENETIC INFORMATION.

RESEARCH OBJECTIVE

To develop a robust and secure hybrid biometric system that integrates fingerprint recognition, DNA analysis, vein pattern recognition, and physiological data analysis to enhance authentication accuracy and reduce vulnerability to spoofing attacks.

PROPOSED SYSTEM

PROPOSED SYSTEM INTRODUCTION

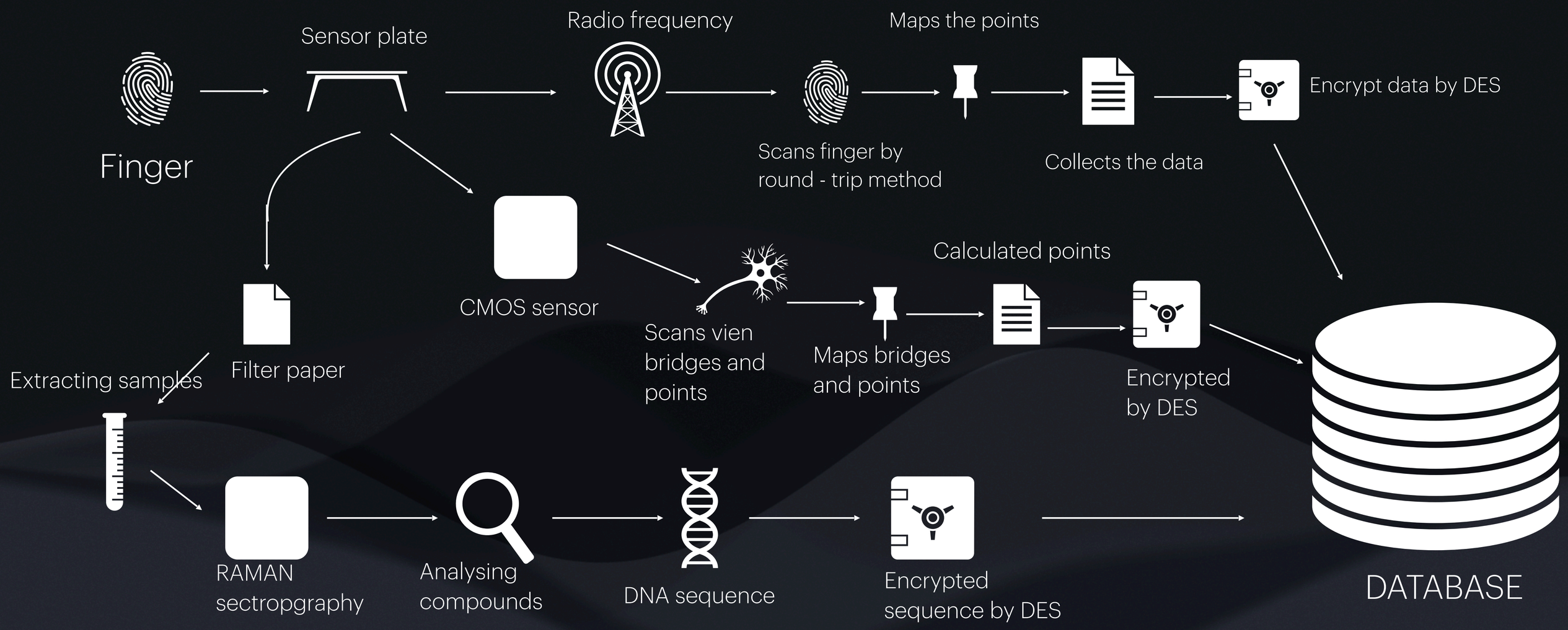
This research proposes a novel hybrid biometric system that leverages the power of multiple biometric modalities to enhance security and reduce the risk of spoofing attacks. By combining fingerprint recognition, DNA analysis, vein pattern recognition, and physiological data analysis, this system offers a robust and reliable solution for personal identification and authentication.

KEY FEATURES

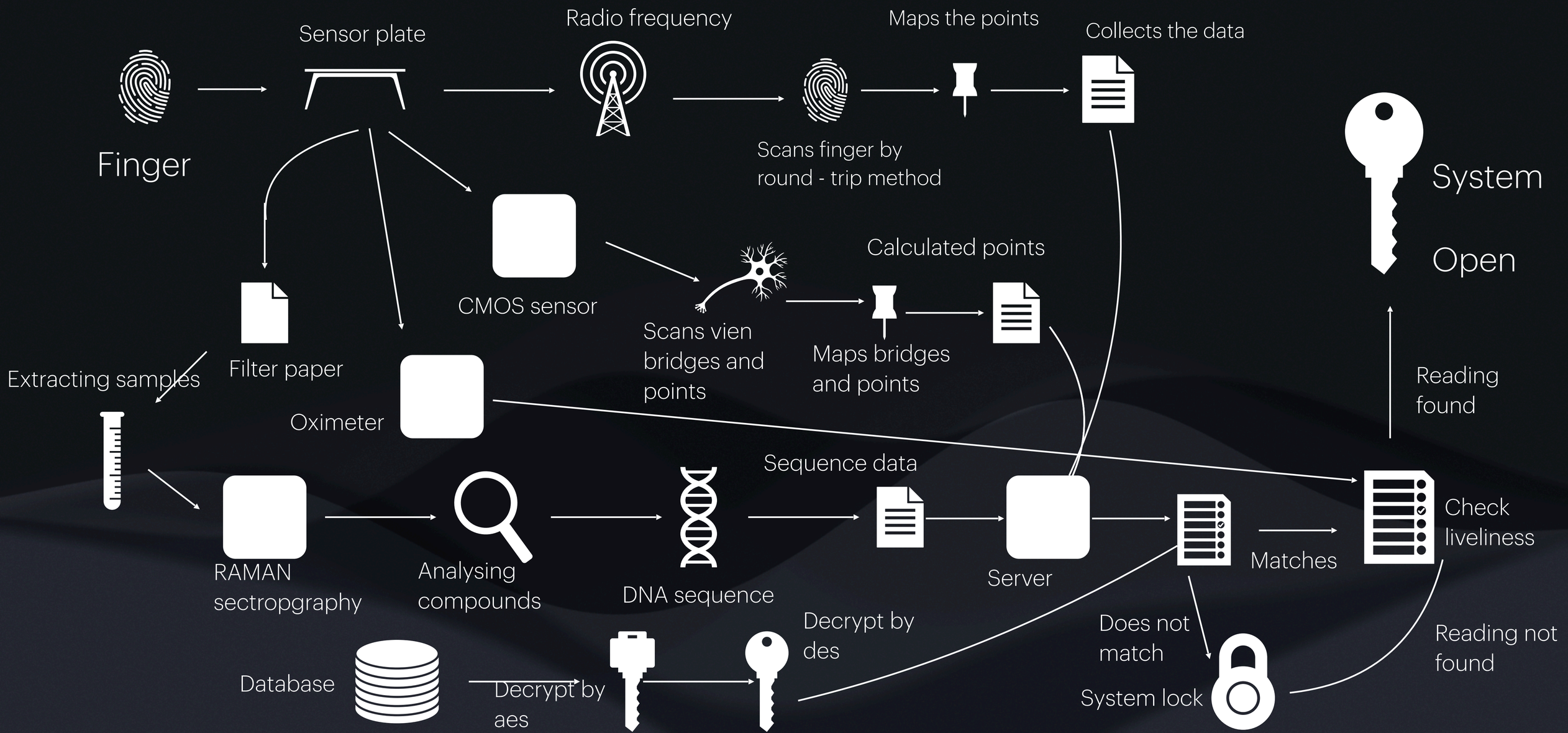
- **Multimodal Biometric Fusion:** Combines multiple biometric modalities to improve accuracy and security.
- **Advanced Sensor Integration:** Utilizes cutting-edge sensors like RF sensors, Raman spectrometers, CMOS sensors, and oximeters for precise data acquisition.
- **Robust Feature Extraction:** Employs efficient algorithms to extract distinctive features from each modality.
- **Effective Feature Fusion:** Leverages advanced fusion techniques to integrate information from different modalities.
- **Secure Authentication:** Implements robust matching and verification algorithms to ensure accurate user identification.
- **Liveness Detection:** Incorporates liveness detection techniques to prevent spoofing attacks.
- **User-Friendly Interface:** Provides a seamless user experience with a user-friendly interface.

This hybrid biometric system has the potential to revolutionize the field of security by providing a highly secure and reliable solution for various applications, including access control, financial transactions, and identity verification.

PROPOSED DIAGRAM - encryption



PROPOSED DIAGRAM - decryption



MODULES IN THE SYSTEM

- Radio frequency module
- CMOS sensor (vien pattern recognition module)
- Oxymeter module
- DNA Collection and Sequencing Module
- Data Encryption and Storage Module
- Matching and Authentication Module
- User Interface (UI) and Control Module

Total Fingerprint Processing Time:

$$T_{fingerprint} = T_{capture} + T_{preprocess} + T_{feature\ extraction} + T_{matching}$$

RF Sensors Module (Proximity and Liveliness Detection)

Function:

- The RF sensor detects the proximity of a person by emitting radio waves and receiving reflected signals from the user. It ensures that the **fingerprint or vein pattern** is from a living person.

Algorithms:

- **Proximity Detection**: The RF sensor uses **signal strength analysis** to determine the distance of an object from the sensor. The **Received Signal Strength Indicator (RSSI)** is used to estimate distance.

RSSI Algorithm:

$$RSSI = P_{tx} - 10n\log_{10}(d) + X_n$$

- Ptx is the transmitted power
- n is the path loss exponent,
- d is the distance from the sensor
- X is a random variable for noise.

- **Liveliness Detection**: RF sensors can be combined with pulse-rate analysis or heat detection algorithms to verify whether the person is alive or using a fake fingerprint.
  - **Pulse Detection**: Use infrared sensing to detect **blood flow** (through a small sensor that detects slight variations in heat or pulse) as a sign of liveness.

Fingerprint Recognition Module

Function:

- Captures **fingerprint features** (such as minutiae) and processes them to create a biometric template for comparison.

Algorithms:

- Image Acquisition and Preprocessing:
  - Fingerprint Image Capture: Captured using a capacitive or optical sensor.
  - **Preprocessing Algorithm**: Includes **image enhancement** (e.g., histogram equalization) to improve the quality of the fingerprint image.
    - **Histogram Equalization**: Adjusts image contrast to improve the visibility of fingerprint features.

Feature Extraction:

- **Minutiae Detection**: Identifies key features such as **ridge endings**, **bifurcations**, and **crossovers**.
- **Pattern Matching**: After feature extraction, algorithms match the template by comparing the fingerprint features with those stored in the database.
  - **Ridge Flow Algorithm**: Detects the flow of ridges in the fingerprint, and minutiae are extracted at **ridge endings** and **bifurcations**.

Matching Algorithm:

- **Minutiae Matching**: Uses the **Euclidean distance** between points to measure the similarity between two sets of minutiae.
- **Fingerprint Matching Score**: Based on the number of matched minutiae, the matching score is calculated using an algorithm such as **Booth's Algorithm** or **N-Nearest Neighbor (NNN)** for determining the match.
  - **Booth's Algorithm**: Efficient matching of minutiae templates in O(log n) time.

Oximeter (Liveliness Detection)

Function:

- Ensures the **liveness** of the user by detecting **blood oxygen saturation** (SpO2) and **pulse rate** to verify that the finger is coming from a living person, not a prosthetic.

Algorithms:

- Pulse Detection:
  - **Photoplethysmography (PPG)** is used to measure **changes in light absorption** by the skin that occurs with each pulse.
  - **Signal Filtering**: The raw PPG signal is filtered to remove noise and isolate the pulse signal. Algorithms such as **Bandpass filters** or **Butterworth filters** can be used.

Algorithm for calculating **SpO2**:

$$\text{SpO2} = \frac{AC_{red}}{AC_{infrared}}$$

Kalman Filter: Used to smooth and process the fluctuating PPG signals to estimate SpO2 levels.

CMOS module (Vein Pattern Recognition Module)

Function:

The **vein pattern recognition** module utilizes **infrared (IR) light** to capture the unique vein patterns in the finger, which is then processed for authentication. The system uses a **CMOS (Complementary Metal-Oxide-Semiconductor)** sensor for capturing high-resolution infrared images of the veins, providing an accurate and non-intrusive method for biometric verification.

Algorithms:

- Image Capture and Preprocessing:
  - **Infrared Imaging**: Capture vein patterns using **infrared (IR)** light, which penetrates the skin and illuminates the veins.
  - **Preprocessing Algorithm**: Removes noise from the infrared image using techniques like **Gaussian blur** and **edge detection** (e.g., **Canny Edge Detection**) to highlight the veins.

Edge Detection Algorithm:

- Canny edge detection:

$$G=\sqrt{(Ix)^2+(Iy)^2}$$

- Feature Extraction:

- **Vein Pattern Matching**: Use **template matching** or **feature-based methods** to extract vein pattern features (such as endpoints, bifurcations, and intersections).

  - **Minutiae Extraction**: Similar to fingerprinting, vein patterns are processed using **minutiae-based** algorithms to extract features for matching.

Matching Algorithm:

- **Dynamic Time Warping (DTW)** or **Euclidean distance** is often used to compare two sets of vein patterns, determining similarity.
  - **DTW Algorithm**: Computes the optimal alignment of two sequences of vein pattern data.

$$\underline{\mathbf{DTW(i,j)= min[DTW(I-1,j),DTW(i,j-1),DTW(i-1,j-1)]+d(i,j)}}$$

DNA Collection and Sequencing Module

Function:

- Extracts **DNA** from sweat or skin cells, sequences it, and creates a DNA template for biometric verification.

Algorithms:

- DNA Extraction:
  - **Sample Preparation**: Extracting DNA from **sweat or skin cells** using **microfluidic devices** or **centrifugation** to separate DNA from the cellular debris.
  - **Lysis**: Chemical agents (such as **detergents** or **enzymes**) break open cells and release DNA.

DNA Sequencing:

- Next-Generation Sequencing (NGS) or Nanopore Sequencing algorithms are used to sequence DNA.
  - **K-mer Analysis**: In NGS, DNA is fragmented, and **K-mer** algorithms (sequence segments) are used to assemble the fragments.
  - **Error Correction Algorithm**: Nanopore sequencing is prone to errors, so algorithms like **MinION error-correction** are applied.

DNA Matching:

- **Sequence Alignment Algorithms**: **Smith-Waterman** or **Needleman-Wunsch** are used for aligning DNA sequences and comparing the query sequence with stored sequences in the database.
  - **Smith-Waterman Algorithm**: A dynamic programming algorithm used for sequence comparison to find the optimal local alignment between two sequences.

$$\underline{\mathbf{Score = max(0, Score(i-1, j-1) + \delta(i,j), Score(i-1, j), Score(i, j-1)).}}$$

Data Encryption and Storage Module

Function:

- Encrypts biometric data (RF, vein, fingerprint, DNA) for security and stores it in the database for later comparison.

Algorithms:

- **DES (Data Encryption Standard)** and **AES (Advanced Encryption Standard)** algorithms are used to protect sensitive biometric data during storage and transmission.
  - **AES**: A symmetric block cipher that encrypts data in blocks of 128 bits using keys of 128, 192, or 256 bits.

Encryption Algorithm for DES and AES:

$$C = E_{DES}(P, K)$$

$$C = E_{AES}(P, K)$$

- Database Storage:
  - **Encrypted biometric templates** are stored in a secure database. Access control is implemented to ensure only authorized systems can decrypt and access the data.

Matching and Authentication Module

Function:

- Decrypts the stored biometric data and compares it with the live data to authenticate the user.

Algorithms:

- **Decryption**: The system decrypts the stored data using **AES** or **DES** algorithms.
  - Decryption Algorithm:

$$P = D_{AES}(C, K)$$

Matching:

- **Fingerprint Matching**: Compares the captured fingerprint minutiae with the stored template using the **Euclidean distance** or **N-Nearest Neighbor** algorithm.
- **Vein Pattern Matching**: Uses **Dynamic Time Warping (DTW)** or **Euclidean distance** to compare vein patterns.
- **DNA Matching**: Uses **sequence alignment algorithms** (e.g., **Smith-Waterman**) to compare the DNA sequence.

ID: Unique identifier for each dataset entry.

Minutiae Points: Number of unique points of ridge characteristics on the fingerprint.

Ridge Density: Measure of ridge thickness or density on the fingerprint (approximate value).

Vein Bifurcations: Number of bifurcation points in the vein pattern (detected by the CMOS sensor).

DNA Sequence: Short DNA sequence fragment extracted from sweat for identification.

Sequence Match Score: Percentage similarity score for the DNA sequence alignment against a reference.

SpO2: Blood oxygen level (liveness check, typically above 95% indicates alive).

Pulse Rate: Heart rate in beats per minute (bpm).

Liveness Check: Result of the liveness test (Pass/Fail).

Processing Time: Total time taken for all module operations in seconds (estimated or simulated based on typical processing times).

Individual Module Processing Time

1. Individual Module Processing Time

Fingerprint Recognition Processing Time

1. **Data Capture**: Time taken to capture the fingerprint image.
2. **Preprocessing**: Time taken for noise reduction, contrast enhancement, and ridge detection.
3. **Feature Extraction**: Time taken to detect minutiae points and create a feature vector.
4. **Matching**: Time taken to compare the feature vector with stored templates.
   Total Fingerprint Processing Time:

$$T_{fingerprint} = T_{capture} + T_{preprocess} + T_{feature\ extraction} + T_{matching}$$

Vein Pattern Recognition Processing Time

1. **Data Capture (CMOS Sensor)**: Time taken by the infrared system to capture the vein pattern.

2. **Preprocessing**: Noise filtering, edge detection, and contrast enhancement.

3. **Feature Extraction**: Detection of bifurcations, endpoints, and vein thickness.

4. **Matching**: Time taken to compare extracted features to stored vein patterns.
   Total Vein Recognition Processing Time:

$$T_{vein} = T_{capture} + T_{preprocess} + T_{feature\ extraction} + T_{matching}$$

DNA Sequence Matching Processing Time

1. **Data Capture**: Time taken to collect a sweat sample from the sensor plate.

2. **DNA Extraction**: Time taken for the chemical or physical process to isolate DNA.

3. **Sequencing**: Time taken for nanopore or other sequencing technology to read a portion of the DNA.

4. **Feature Extraction and Matching**: Time to process and match the sequence to a stored reference.

   Total DNA Processing Time:

$$T_{DNA} = T_{capture} + T_{extraction} + T_{sequencing} + T_{matching}$$

Oximeter (Liveness Detection) Processing Time

1. Data Capture: Time taken to measure blood oxygen levels and pulse rate.
2. Processing and Threshold Checking: Time to compare the captured data with predefined thresholds for liveness.

$$T_{liveness} = T_{capture} + T_{processing}$$

In a sequential system, the total processing time is the sum of each module's processing time

$$T_{total_sequential} = T_{fingerprint} + T_{vein} + T_{DNA} + T_{liveness}$$

If the modules run concurrently, the total processing time will be approximately the time of the **slowest module**:

$$T_{total_concurrent} = \max(T_{fingerprin}, T_{vein}, T_{DNA}, T_{liveness})$$

The **Weighted Security Strength Calculation** in a hybrid biometric system considers the security contributions of each biometric modality (fingerprint, vein recognition, DNA sequence, and liveness detection) based on their individual reliability and resistance to spoofing or attacks. By assigning weights to each modality, we can calculate an overall security strength score that reflects the combined robustness of the system.

WEIGHT FOR FINGERPRINT = 0.25

WEIGHT FOR VEIN = 0.25

WEIGHT FOR DNA = 0.4

WEIGHT FOR LIVELINESS = 0.1

Weighted Security Strength Formula

$$S_{total} = (W_{fingerprint} * S_{fingerprint}) + (W_{vein} * S_{vein}) + (W_{DNA} * S_{DNA}) + (W_{liveness} * S_{liveness})$$

SECURITY STRENGTH FOR FINGERPRINT - 85

SECURITY STRENGTH FOR VEIN - 90

SECURITY STRENGTH FOR DNA - 98

SECURITY STRENGTH FOR LIVENESS - 80

TOTAL SECURITY STRENGTH = (0.25×85)+(0.25×90)+(0.4×98)+(0.1×80)

Calculating each term:

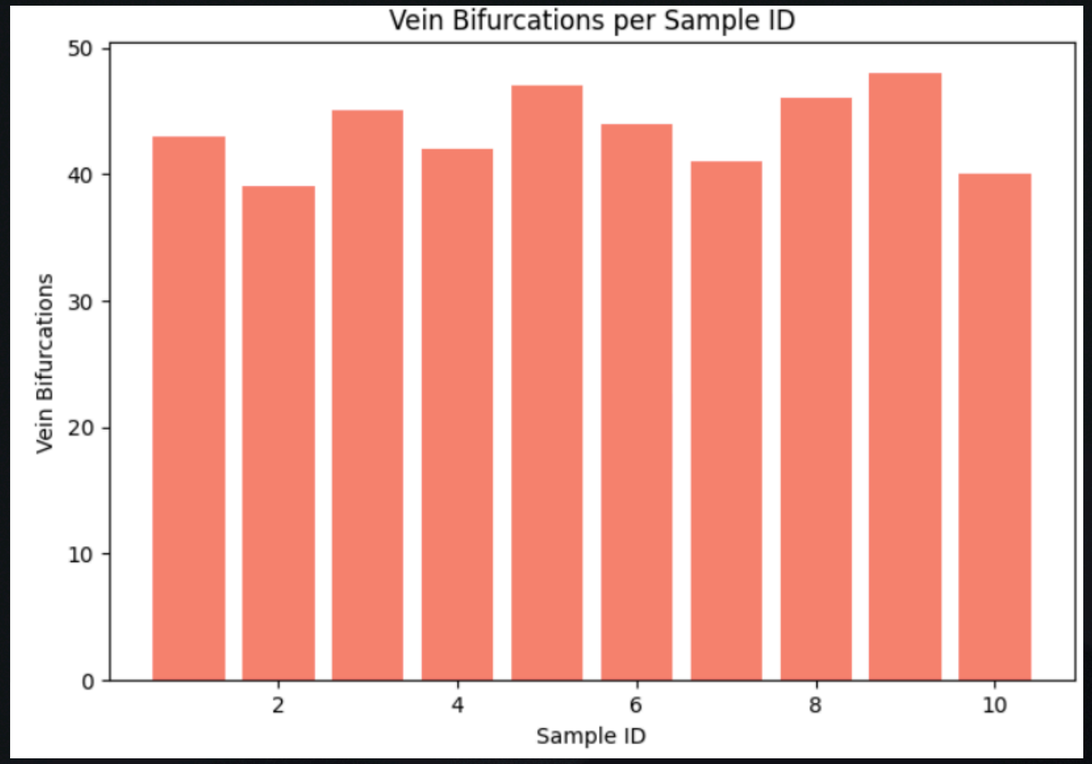
- Fingerprint: $0.25 \times 85 = 21.25$
- Vein: $0.25 \times 90 = 22.5$
- DNA: $0.4 \times 98 = 39.2$
- Liveness: 0.1×80=8.0

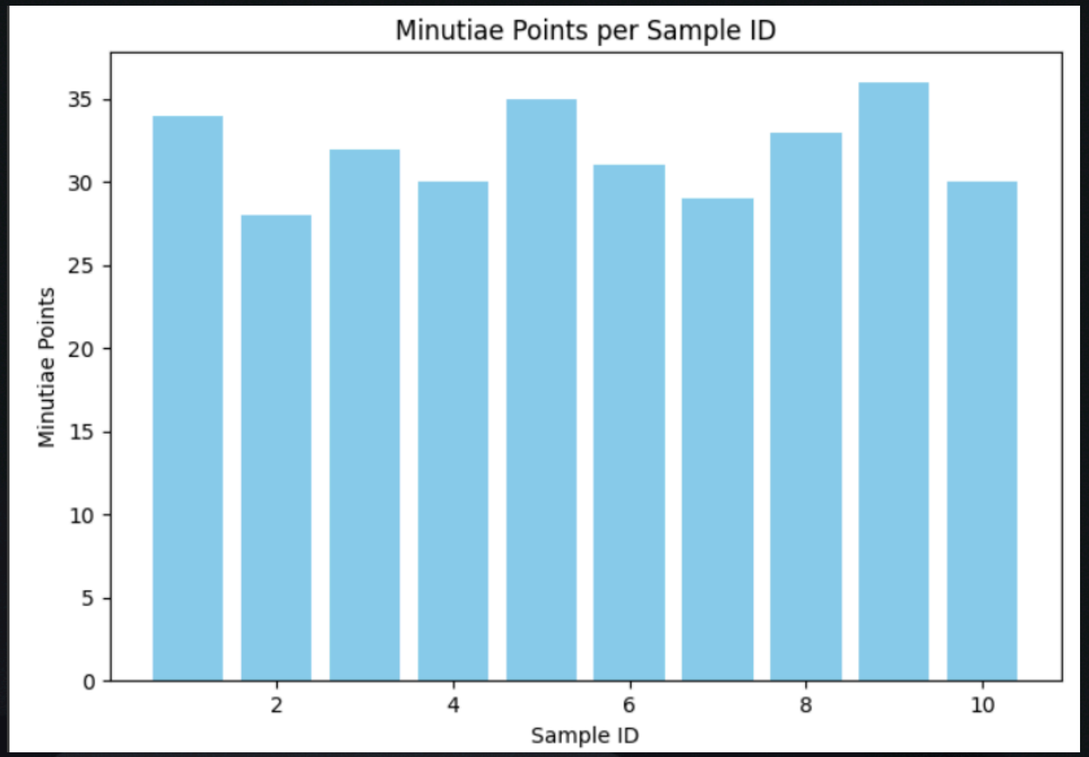
  TOTAL SECURITY STRENGTH = 21.25+22.5+39.2+8.0 = 90.95

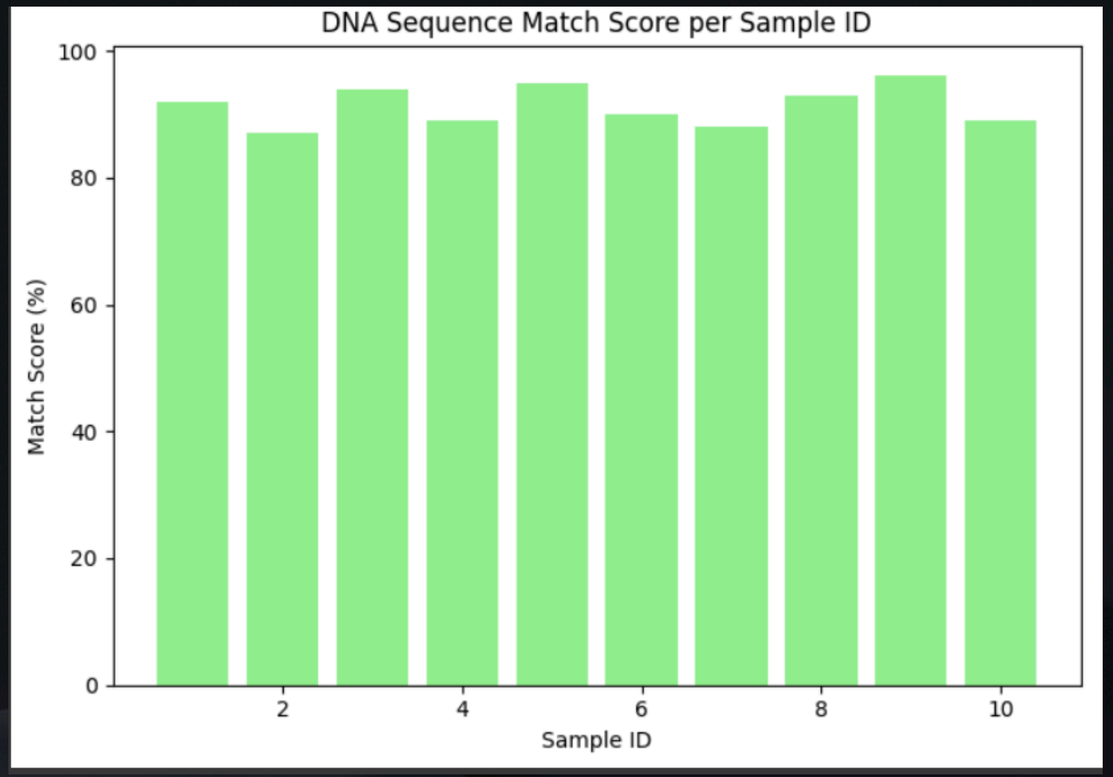
- So the **overall security strength** of this hybrid biometric system is approximately **90.95 out of 100**.

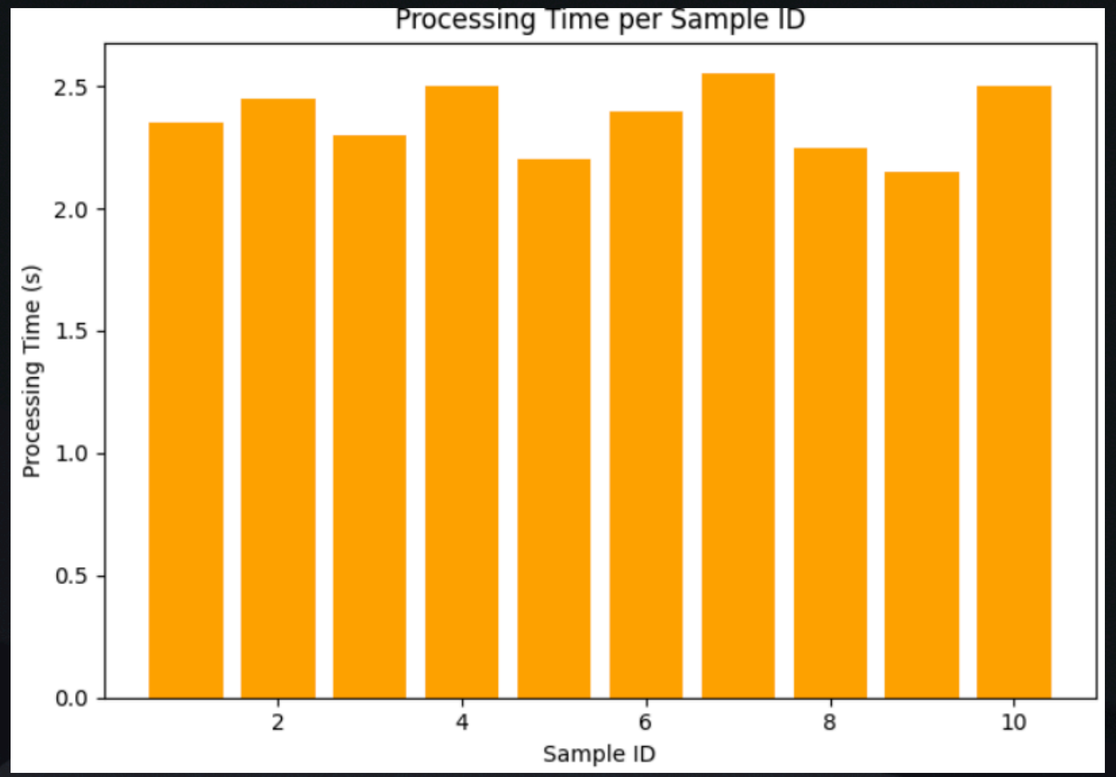
DATASET

ID	Minutiae Points	Ridge Density	Vein Bifurcations	DNA Sequence	Sequence Match Score	SpO2	Pulse Rate	Liveness Check	Processing Time (s)
1	34	8.1	43	ATCGGCTA	92%	98%	72 bpm	Pass	2.35
2	28	7.8	39	GATCGGAT	87%	97%	69 bpm	Pass	2.45
3	32	8.3	45	CGTAGCTA	94%	96%	74 bpm	Pass	2.3
4	30	8	42	ATGCCGTA	89%	98%	75 bpm	Pass	2.5
5	35	7.9	47	TCGATCGA	95%	99%	70 bpm	Pass	2.2
6	31	8.2	44	CGTAGGTC	90%	95%	71 bpm	Pass	2.4
7	29	8	41	ATCGTACG	88%	96%	68 bpm	Pass	2.55
8	33	8.4	46	GCTAGCTA	93%	97%	73 bpm	Pass	2.25
9	36	8.1	48	ATCGGCTG	96%	98%	69 bpm	Pass	2.15
10	30	7.7	40	CGTACGAT	89%	97%	74 bpm	Pass	2.5


Vein Bifurcations per Sample ID
Vein Bifurcations
Sample ID
0
10
20
30
40
50
2
4
6
8
10


Minutiae Points per Sample ID
Minutiae Points
Sample ID
0
5
10
15
20
25
30
35
2
4
6
8
10


DNA Sequence Match Score per Sample ID
Match Score (%)
100
80
60
40
20
0
2
4
6
8
10
Sample ID


Processing Time per Sample ID
Processing Time (s)
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
8
10
Sample ID

Code

Extraction

CODE:

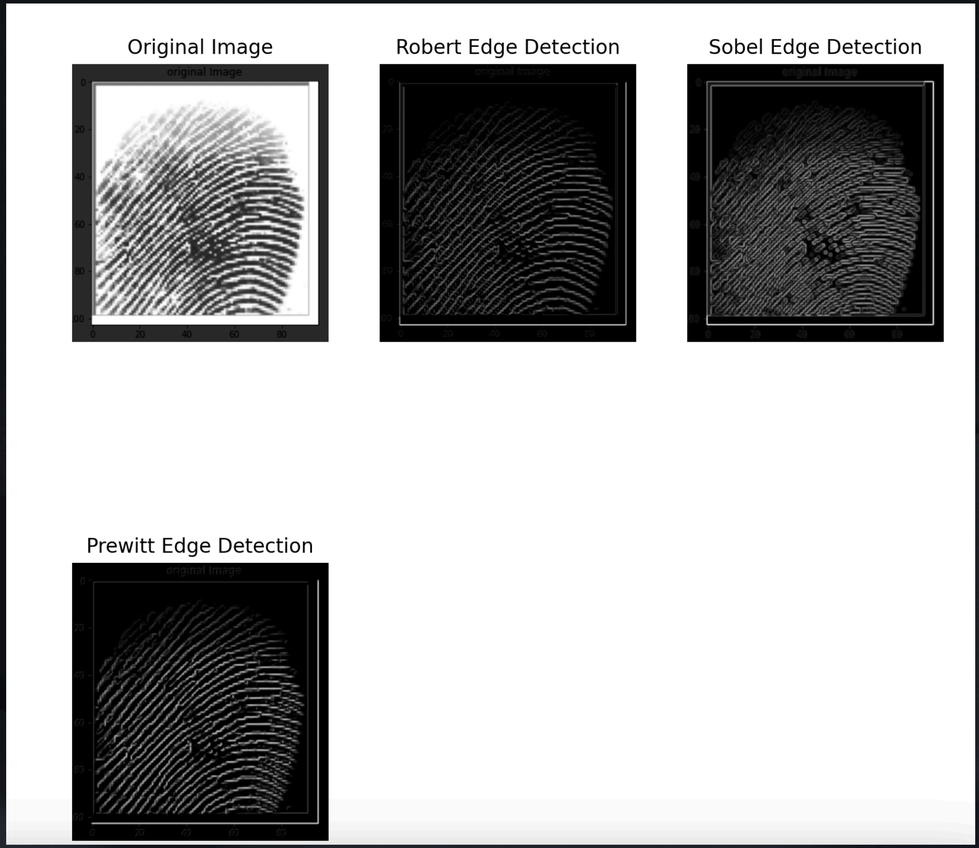
Edge cutting

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image (replace this with the correct path to your image)
image_path = '/Users/nirudeeswar/Desktop/fingerprint.png'  # Update with the actual path
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image is loaded correctly
if image is None:
    print(f"Failed to load the image at: {image_path}")
else:
    print(f"Image loaded successfully from: {image_path}")

    # Step 1: Gaussian Blurring to remove noise
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # ------------------------ Robert Filter ------------------------
    # Robert operator kernels for edge detection
    robert_kernel_x = np.array([[1, 0], [0, -1]], dtype=float)
    robert_kernel_y = np.array([[0, 1], [-1, 0]], dtype=float)

    # Apply Robert filter (convolve the image with the kernels)
    robert_x = cv2.filter2D(blurred_image, -1, robert_kernel_x)
    robert_y = cv2.filter2D(blurred_image, -1, robert_kernel_y)

    # Convert to float64 to avoid errors with magnitude
    robert_x = np.float64(robert_x)
    robert_y = np.float64(robert_y)

    # Compute magnitude of the gradient
    robert_edges = cv2.magnitude(robert_x, robert_y)

    # ------------------------ Sobel Filter ------------------------
    # Sobel operator kernels for edge detection
    sobel_x = cv2.Sobel(blurred_image, cv2.CV_64F, 1, 0, ksize=3)  # Sobel in X direction
    sobel_y = cv2.Sobel(blurred_image, cv2.CV_64F, 0, 1, ksize=3)  # Sobel in Y direction
    sobel_edges = cv2.magnitude(sobel_x, sobel_y)  # Combine X and Y edges

    # ------------------------ Prewitt Filter ----------------------
    # Prewitt operator kernels for edge detection
    prewitt_kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=float)
    prewitt_kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=float)

    # Apply Prewitt filter (convolve the image with the kernels)
    prewitt_x = cv2.filter2D(blurred_image, -1, prewitt_kernel_x)
    prewitt_y = cv2.filter2D(blurred_image, -1, prewitt_kernel_y)

    # Convert to float64 to avoid errors with magnitude
    prewitt_x = np.float64(prewitt_x)
    prewitt_y = np.float64(prewitt_y)

    # Compute magnitude of the gradient
    prewitt_edges = cv2.magnitude(prewitt_x, prewitt_y)
```

```python
# ------------------------ Display Results ---------------------
plt.figure(figsize=(15, 10))

# Show the original image
plt.subplot(2, 3, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')

# Show the Robert edges
plt.subplot(2, 3, 2)
plt.title("Robert Edge Detection")
plt.imshow(robert_edges, cmap='gray')
plt.axis('off')

# Show the Sobel edges
plt.subplot(2, 3, 3)
plt.title("Sobel Edge Detection")
plt.imshow(sobel_edges, cmap='gray')
plt.axis('off')

# Show the Prewitt edges
plt.subplot(2, 3, 4)
plt.title("Prewitt Edge Detection")
plt.imshow(prewitt_edges, cmap='gray')
plt.axis('off')

plt.show()
```

- Input:

Output

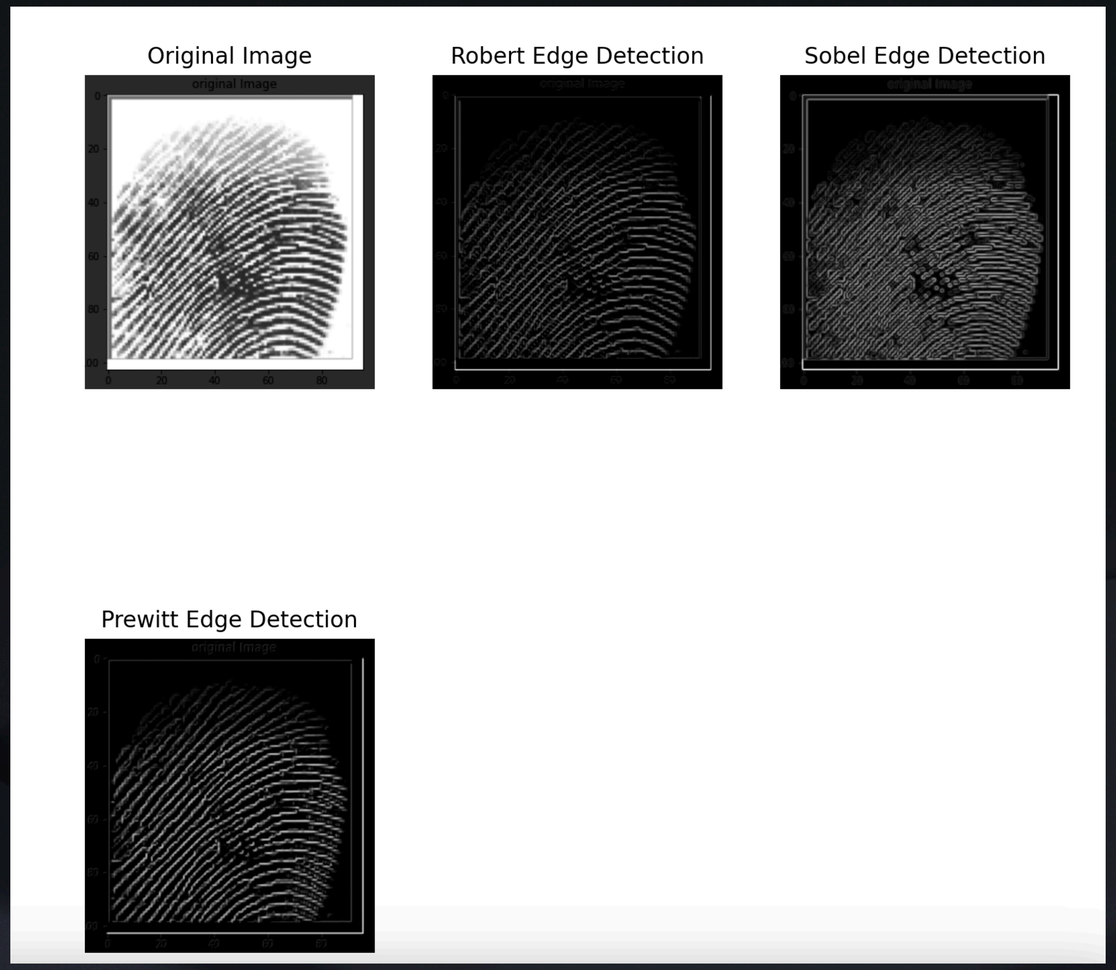

Original Image
Robert Edge Detection
Sobel Edge Detection
Prewitt Edge Detection

Termination and Bifurcation

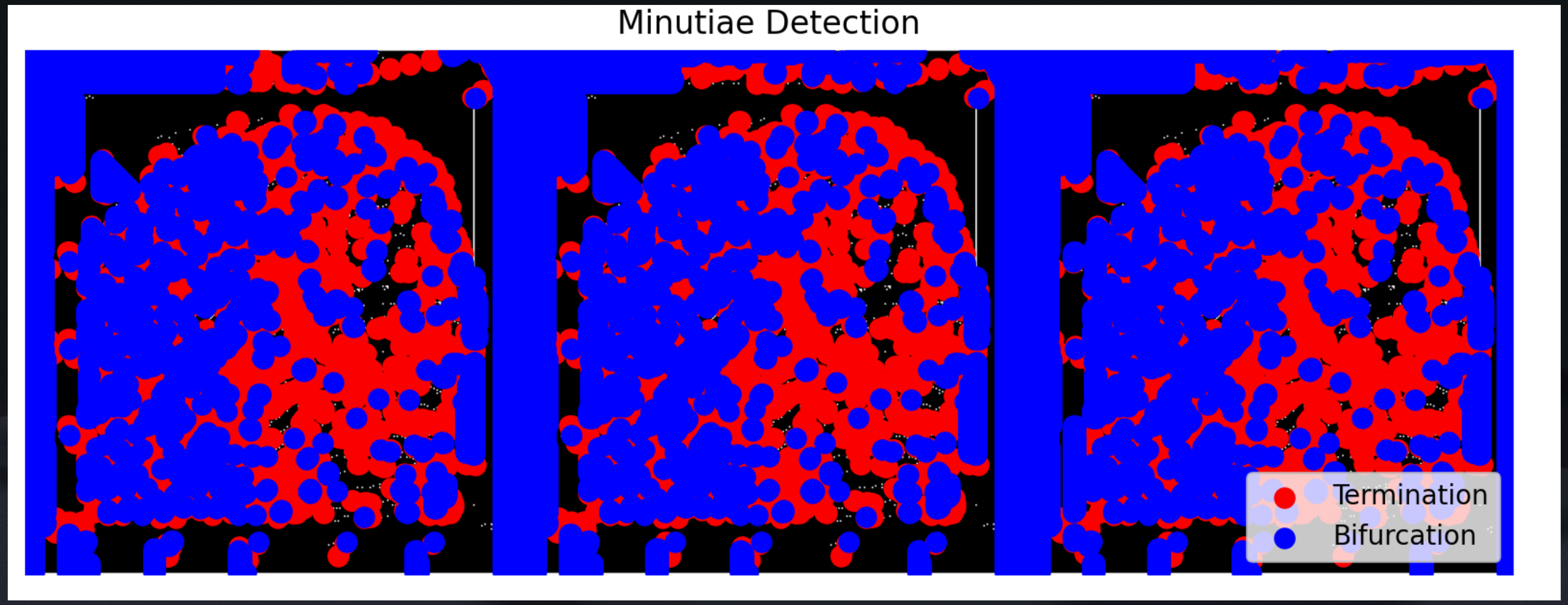
```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os

# Improved Preprocessing: Contrast and Noise Removal
def preprocess_image(image):
    # Apply Gaussian Blurring to reduce noise
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply Histogram Equalization to improve contrast
    equalized_image = cv2.equalizeHist(blurred_image)

    # Apply Adaptive Thresholding for better binarization in varying light conditions
    binary_image = cv2.adaptiveThreshold(equalized_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                         cv2.THRESH_BINARY, 11, 2)
    return binary_image

# Zhang-Suen Thinning Algorithm (fixed)
def zhang_suen_thinning(image):
    img = image.copy()
    prev = np.zeros(image.shape, np.uint8)

    while True:
        marker = np.zeros(img.shape, np.uint8)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                if img[i, j] == 255:
                    neighbors = img[i-1:i+2, j-1:j+2].flatten()
                    neighbors = np.delete(neighbors, 4)  # Remove center pixel
                    ridge_neighbors = np.sum(neighbors == 255)
                    if ridge_neighbors >= 2 and ridge_neighbors <= 6:
                        transitions = np.sum(np.abs(np.diff(neighbors))) // 255
                        if transitions == 1:
                            marker[i, j] = 255
        img = cv2.subtract(img, marker)

        marker = np.zeros(img.shape, np.uint8)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                if img[i, j] == 255:
                    neighbors = img[i-1:i+2, j-1:j+2].flatten()
                    neighbors = np.delete(neighbors, 4)
                    ridge_neighbors = np.sum(neighbors == 255)
                    if ridge_neighbors >= 2 and ridge_neighbors <= 6:
                        transitions = np.sum(np.abs(np.diff(neighbors))) // 255
                        if transitions == 1:
                            marker[i, j] = 255
        img = cv2.subtract(img, marker)

        if np.array_equal(img, prev):
            break
        prev = img.copy()

    return img
```

```python
# Function to check if image exists
def load_image(image_path):
    if not os.path.exists(image_path):
        print(f"Image not found at: {image_path}")
        return None
    return cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Manually specify the image path or select file
image_path = '/Users/nirudeeswar/Desktop/fingerprintextraction.png'  # Adjust this path based on your image location
image = load_image(image_path)

if image is None:
    print("Failed to load the image. Please check the file path.")
else:
    print("Image loaded successfully.")

    # Step 1: Preprocess Image (Gaussian Blur, Histogram Equalization, and Adaptive Thresholding)
    preprocessed_image = preprocess_image(image)

    # Step 2: Apply Zhang-Suen Thinning
    thinned_image = zhang_suen_thinning(preprocessed_image)

    # Step 3: Extract minutiae (terminations and bifurcations)
    minutiae = extract_minutiae(thinned_image)

    # Display the results
    plt.figure(figsize=(10, 10))
    plt.imshow(thinned_image, cmap='gray')

    # Extract x and y coordinates from minutiae points for termination and bifurcation
    terminations_x, terminations_y = zip(*minutiae['terminations']) if minutiae['terminations'] else ([], [])
    bifurcations_x, bifurcations_y = zip(*minutiae['bifurcations']) if minutiae['bifurcations'] else ([], [])

    # Plot termination points in red and bifurcation points in blue
    plt.scatter(terminations_x, terminations_y, color='red', label='Termination', s=50)
    plt.scatter(bifurcations_x, bifurcations_y, color='blue', label='Bifurcation', s=50)

    # Add legend and title
    plt.legend()
    plt.title("Minutiae Detection")
    plt.axis('off')
    plt.show()
```

Input


Original Image
Robert Edge Detection
Sobel Edge Detection
Prewitt Edge Detection

Output



Converting the data to binary

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os

# Improved Preprocessing: Contrast and Noise Removal
def preprocess_image(image):
    # Apply Gaussian Blurring to reduce noise
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply Histogram Equalization to improve contrast
    equalized_image = cv2.equalizeHist(blurred_image)

    # Apply Adaptive Thresholding for better binarization in varying light conditions
    binary_image = cv2.adaptiveThreshold(equalized_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                         cv2.THRESH_BINARY, 11, 2)
    return binary_image

# Zhang-Suen Thinning Algorithm (fixed)
def zhang_suen_thinning(image):
    img = image.copy()
    prev = np.zeros(image.shape, np.uint8)

    while True:
        marker = np.zeros(img.shape, np.uint8)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                if img[i, j] == 255:
                    neighbors = img[i-1:i+2, j-1:j+2].flatten()
                    neighbors = np.delete(neighbors, 4)  # Remove center pixel
                    ridge_neighbors = np.sum(neighbors == 255)
                    if ridge_neighbors >= 2 and ridge_neighbors <= 6:
                        transitions = np.sum(np.abs(np.diff(neighbors))) // 255
                        if transitions == 1:
                            marker[i, j] = 255
        img = cv2.subtract(img, marker)

        marker = np.zeros(img.shape, np.uint8)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                if img[i, j] == 255:
                    neighbors = img[i-1:i+2, j-1:j+2].flatten()
                    neighbors = np.delete(neighbors, 4)
                    ridge_neighbors = np.sum(neighbors == 255)
                    if ridge_neighbors >= 2 and ridge_neighbors <= 6:
                        transitions = np.sum(np.abs(np.diff(neighbors))) // 255
                        if transitions == 1:
                            marker[i, j] = 255
        img = cv2.subtract(img, marker)

        if np.array_equal(img, prev):
            break
        prev = img.copy()

    return img
```

```python
# Function to check if image exists
def load_image(image_path):
    if not os.path.exists(image_path):
        print(f"Image not found at: {image_path}")
        return None
    return cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Manually specify the image path or select file
image_path = '/Users/nirudeeswar/Desktop/fingerprintextraction.png'  # Adjust this path based on your image location
image = load_image(image_path)

if image is None:
    print("Failed to load the image. Please check the file path.")
else:
    print("Image loaded successfully.")

    # Step 1: Preprocess Image (Gaussian Blur, Histogram Equalization, and Adaptive Thresholding)
    preprocessed_image = preprocess_image(image)

    # Step 2: Apply Zhang-Suen Thinning
    thinned_image = zhang_suen_thinning(preprocessed_image)

    # Step 3: Extract minutiae (terminations and bifurcations)
    minutiae = extract_minutiae(thinned_image)

    # Display the results
    plt.figure(figsize=(10, 10))
    plt.imshow(thinned_image, cmap='gray')

    # Extract x and y coordinates from minutiae points for termination and bifurcation
    terminations_x, terminations_y = zip(*minutiae['terminations']) if minutiae['terminations'] else ([], [])
    bifurcations_x, bifurcations_y = zip(*minutiae['bifurcations']) if minutiae['bifurcations'] else ([], [])

    # Plot termination points in red and bifurcation points in blue
    plt.scatter(terminations_x, terminations_y, color='red', label='Termination', s=50)
    plt.scatter(bifurcations_x, bifurcations_y, color='blue', label='Bifurcation', s=50)

    # Add legend and title
    plt.legend()
    plt.title("Minutiae Detection")
    plt.axis('off')
    plt.show()
```

Output

```
01000000110100101000000000000000010
01000000110111001100000000000000011
01000000101111010100000000000000100
01000000110010000100000000000000100
01000000110010001000000000000000100
01000000110010101100000000000000100
01000000110110000000000000000000100
01000000100111110100000000000000101
01000000100111111000000000000000101
01000000101111110000000000000000101
01000000110000010100000000000000101
01000000110001100000000000000000101
01000000110010010100000000000000101
01000000110101111100000000000000101
01000000111100000000000000000000101
01000000100010010110000000000000101
01000000101111011000000000000000110
01000000101111101100000000000000110
01000000110110000100000000000000110
01000000110111010000000000000000110
01000000111100000000000000000000110
01000000100010010110000000000000110
01000000101101010000000000000000111
01000000110000011000000000000000111
01000000110100110000000000000000111
01000000110110000000000000000000111
01000000110111101000000000000000111
01000000111000000000000000000000111
01000000110000011100000000000000111
01000000101001110100000000000001000
01000000110010010000000000000001000
01000000110010110100000000000001000
01000000110100010000000000000001000
01000000110100010000000000000001000
01000000110100100100000000000001000
01000000110110001000000000000001000
01000000110111010100000000000001000
01000000110111101000000000000001000
01000000100001111100000000000001000
01000000101100101100000000000001001
01000000101101000100000000000001001
01000000110001101100000000000001001
01000000110100100100000000000001001
01000000110110001000000000000001001
01000000110111101000000000000001001
01000000111000100100000000000001001
01000000111000101000000000000001001
01000000100001111100000000000001001
01000000110000011110000000000001001
01000000101001111000000000000001010
01000000101010101000000000000001010
01000000110100001100000000000001010
01000000111001100000000000000001010
01000000111001100100000000000001010
01000000111001111100000000000001010
01000000111010101100000000000001010
01000000101010010000000000000001011
01000000101010011000000000000001011
01000000101111110100000000000001011
01000000101111111000000000000001011
01000000110001111000000000000001011
01000000110011011000000000000001011
01000000110011011100000000000001011
01000000110101001000000000000001011
01000000110101011100000000000001011
01000000110110110000000000000001011
01000000110110110100000000000001011
01000000110111001100000000000001011
01000000111000111100000000000001011
01000000111010011100000000000001011
```

Txt encrypted by des

```python
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad
import os

# Function to verify if the file exists
def verify_file(file_path):
    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return False
    return True

# Function to read binary data from a file
def read_binary_file(file_path):
    with open(file_path, 'rb') as file:
        return file.read()

# Function to encrypt data using DES
def encrypt_des(binary_data, key):
    # Create a DES cipher object with the key
    des_cipher = DES.new(key, DES.MODE_ECB)

    # Pad the data to make it a multiple of 8 bytes
    padded_data = pad(binary_data, DES.block_size)

    # Encrypt the data
    encrypted_data = des_cipher.encrypt(padded_data)
    return encrypted_data

# Function to save encrypted data to a file
def save_encrypted_file(output_file_path, encrypted_data):
    with open(output_file_path, 'wb') as file:
        file.write(encrypted_data)
    print(f"Encrypted data saved at: {output_file_path}")

# File paths
input_file_path = '/Users/nirudeeswar/Desktop/minutiae_binary_data.txt'  # Update this to your file location
output_file_path = '/Users/nirudeeswar/Desktop/encrypted_data.des'

# Key must be 8 bytes long for DES
key = b'secretk1'  # Replace 'secretk1' with your 8-byte key

# Verify if the input file exists
if not verify_file(input_file_path):
    print("Please check the file path and try again.")
else:
    print("File found. Proceeding with encryption.")

    # Step 1: Read binary data from the file
    binary_data = read_binary_file(input_file_path)
    print(f"Binary data read from file: {binary_data}")

    # Step 2: Encrypt the binary data using DES
    encrypted_data = encrypt_des(binary_data, key)
    print(f"Data successfully encrypted.")

    # Step 3: Save the encrypted data to a new file
    save_encrypted_file(output_file_path, encrypted_data)
```

AES

```python
from Crypto.Cipher import DES, AES
from Crypto.Util.Padding import pad, unpad
import os

# Function to verify if the file exists
def verify_file(file_path):
    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return False
    return True

# Function to read binary data from a file
def read_binary_file(file_path):
    with open(file_path, 'rb') as file:
        return file.read()

# Function to save encrypted data to a file
def save_encrypted_file(output_file_path, encrypted_data):
    with open(output_file_path, 'wb') as file:
        file.write(encrypted_data)
    print(f"Encrypted data saved at: {output_file_path}")

# Function to decrypt data using DES
def decrypt_des(encrypted_data, key):
    des_cipher = DES.new(key, DES.MODE_ECB)
    decrypted_data = unpad(des_cipher.decrypt(encrypted_data), DES.block_size)
    return decrypted_data

# Function to encrypt data using AES
def encrypt_aes(data, key):
    aes_cipher = AES.new(key, AES.MODE_ECB)
    padded_data = pad(data, AES.block_size)
    encrypted_data = aes_cipher.encrypt(padded_data)
    return encrypted_data

# File paths
des_encrypted_file_path = '/Users/nirudeeswar/Desktop/encrypted_data.des'  # Adjust the path to your DES-encrypted file
aes_encrypted_file_path = '/Users/nirudeeswar/Desktop/aes_encrypted_data.aes'

# DES and AES keys
des_key = b'secretk1'  # Must be 8 bytes long
aes_key = b'securekey1234567'  # Must be 16, 24, or 32 bytes long
```

```python
# Step 1: Verify if the DES-encrypted file exists
if not verify_file(des_encrypted_file_path):
    print("Please check the DES-encrypted file path and try again.")
else:
    print("DES-encrypted file found. Proceeding with decryption.")

    # Step 2: Read the DES-encrypted file
    des_encrypted_data = read_binary_file(des_encrypted_file_path)
    print(f"DES-encrypted data read successfully.")

    # Step 3: Decrypt the DES-encrypted data
    try:
        decrypted_data = decrypt_des(des_encrypted_data, des_key)
        print(f"Data successfully decrypted from DES.")
    except ValueError as e:
        print(f"Error during DES decryption: {e}")
        exit()

    # Step 4: Encrypt the decrypted data using AES
    try:
        aes_encrypted_data = encrypt_aes(decrypted_data, aes_key)
        print(f"Data successfully encrypted using AES.")
    except ValueError as e:
        print(f"Error during AES encryption: {e}")
        exit()

    # Step 5: Save the AES-encrypted data to a new file
    save_encrypted_file(aes_encrypted_file_path, aes_encrypted_data)
```

Vein pattern

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Set the correct file path to the image
image_path = "/Users/nirudeeswar/Desktop/vein.png"  # Update this path as needed
image = cv2.imread(image_path)

# Check if the image is loaded correctly
if image is None:
    print("Error: Unable to load the image. Please check the file path.")
    exit()

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Enhance contrast using CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_image = clahe.apply(gray_image)

# Apply Gaussian Blur
blurred_image = cv2.GaussianBlur(enhanced_image, (5, 5), 0)

# Perform adaptive thresholding
thresh_image = cv2.adaptiveThreshold(
    blurred_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 3
)

# Skeletonize to get vein patterns
skeleton = np.zeros_like(thresh_image)
element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
temp_image = thresh_image.copy()

while True:
    eroded = cv2.erode(temp_image, element)
    temp_opened = cv2.dilate(eroded, element)
    temp_skeleton = cv2.subtract(temp_image, temp_opened)
    skeleton = cv2.bitwise_or(skeleton, temp_skeleton)
    temp_image = eroded.copy()
    if cv2.countNonZero(temp_image) == 0:
        break

# Display results
plt.figure(figsize=(10, 10))
plt.subplot(1, 4, 1), plt.title("Original"), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 4, 2), plt.title("Grayscale"), plt.imshow(gray_image, cmap="gray")
plt.subplot(1, 4, 3), plt.title("Enhanced & Thresholded"), plt.imshow(thresh_image, cmap="gray")
plt.subplot(1, 4, 4), plt.title("Skeleton (Vein Pattern)"), plt.imshow(skeleton, cmap="gray")
plt.show()
```

Output

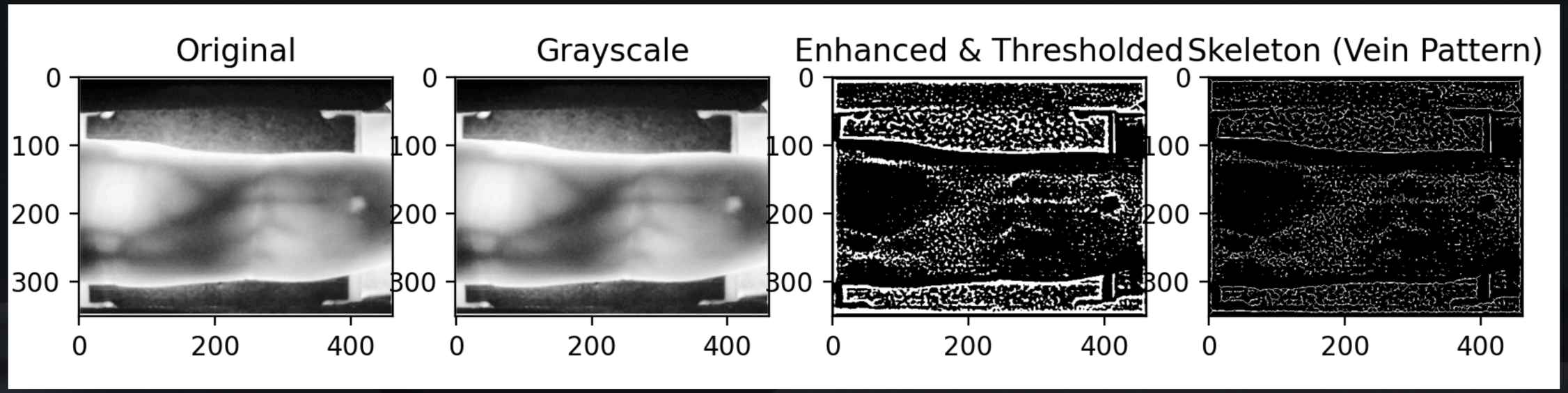

Original
Grayscale
Enhanced & Thresholded
Skeleton (Vein Pattern)
0
100
200
300
0
200
400

Image to data

```python
import cv2
import numpy as np

# Load the image in grayscale
image_path = "/Users/nirudeeswar/Desktop/vein.png"  # Update to your image path
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded correctly
if image is None:
    print(f"Error: Unable to load the image from {image_path}. Check the file path.")
    exit()

# Crop the region of interest (y: 300 to 100, x: 0 to 400)
cropped_image = image[100:300, 0:400]

# Convert to binary (0 and 1) using thresholding
_, binary_image = cv2.threshold(cropped_image, 128, 1, cv2.THRESH_BINARY)

# Save the binary data to a text file
output_file_path = "/Users/nirudeeswar/Desktop/vein_binary_data.txt"  # Update as needed
np.savetxt(output_file_path, binary_image, fmt='%d')

print(f"Binary data saved to {output_file_path}")
```

Output

```
1 1 1 
```

DES

```python
import os

input_file = "/Users/nirudeeswar/Desktop/vein_binary_data.txt"  # Update to the correct path
output_file = "/Users/nirudeeswar/Desktop/vein_binary_data_encrypted.txt"

if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Proceed with the encryption code
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad

key = b'secure12'  # Key must be 8 bytes
with open(input_file, 'rb') as f:
    data = f.read()

cipher = DES.new(key, DES.MODE_ECB)
padded_data = pad(data, DES.block_size)
encrypted_data = cipher.encrypt(padded_data)

with open(output_file, 'wb') as f:
    f.write(encrypted_data)

print(f"File encrypted successfully. Saved to {output_file}")
```

!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8™¨klfilv:¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒ
Â©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸
¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒÂ©$¸¥ÀRÒ
Â©$¸¥ÀRÒÂ©$EŸÀ%;±.À!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!
ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!ŒÛå2éP8!

AES

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import os

# File paths
input_file = "/Users/nirudeeswar/Desktop/vein_binary_data_encrypted.txt"  # Update this path
output_file = "/Users/nirudeeswar/Desktop/vein_binary_data_encrypted_aes.txt"

# AES key (16, 24, or 32 bytes). Ensure it's the correct length.
key = b'mysecurekey12345'  # 16 bytes key
iv = b'initialvector123'  # 16 bytes IV (Initialization Vector)

# Check if the input file exists
if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Read the input file
with open(input_file, 'rb') as f:
    data = f.read()

# Encrypt the data
cipher = AES.new(key, AES.MODE_CBC, iv)
padded_data = pad(data, AES.block_size)  # Pad data to match AES block size (16 bytes)
encrypted_data = cipher.encrypt(padded_data)

# Write the encrypted data to the output file
with open(output_file, 'wb') as f:
    f.write(encrypted_data)

print(f"File encrypted successfully using AES. Saved to {output_file}")
```

```
_                                                                  ¯
]Bë~¿„/iqQ•™Ő˝@W∏Q_g∞ïñ·ñÆ˙“T;æ∫û#eûìõ
°/hπŒ¸s“[N;r∫ú+'„1X“æX£qE`ß>=\Ê™‡#tê˝uñäSj5◆!ßÿOhÓ4p
ìáºi'ÎΩô’°H+Ñÿ≠5¿©HÙÉπiæ◆wXzksq§ŏU∫˙8ıFU‹qmQ˜jÄW<QÑ`2T¿o#\^A∫(BæYv°‡ãe·◆i∏ußtÇ[Ï´£YÅ≤Ó"·+Bñ˜°f€u@M¥T”LÅG=£ë«{Ô/?9j_ÛÊzæ;<'ÃÀ£P$fl¸ÇZÄË~5ò˝…ly¬1wDæõä+2%T…bYW±Rk…Ëπ√–Õ‘∞áIî°Õà}jπ`ér"Í.#E∏JáKlRΩ÷l◆Á7◆±,$(æ3◆=4∞≤º˜pË™h/
îïéÚ˜˜Ëbû≠∑ÜoB(ə/fi¥Ôyi®látÇãπÅø¥°√'h¢£C7Ÿi
ÆÊÜ58C@Ő•#Åplvë(◆6)1*C„/{–Ùßñ˙Ââg≤iä¨UúMW=a\œ|≠Ì∫d•„/:,#'}–ÁÔ≤ÿm±âÀ®ôà.ix¸°Í:|
K8s€\Æ®ÁÚ
1v`»*´m
üU,°Q˜s∫áÃèÀ =iØ∑˚„%ö&
M/vUûrJxAÔùÜ]ZêxYv–à≥◆eù•/əÕÀyü^AMbJ,'FdfõVÉ•ΩsəDB
·pã_ás ∏@pHg§–IF¸á*Ì_åÏi;«–…±
Î1HõFu†A¸ÿøòßñŸ– r˙)aPgä“ÜAÒÀÎ)Ô•®ƒ™ö^`ªø¶”t†6flgn◊bh–BΔªwê≈
1∑
Pur÷ÿîRX%P„Ç)ò˘“wY5_^=v‡´%Ò@π&Ø√rC§&èG´öP%◊j`DlÜéMQzπÙπV:ò˙êz–Nme◊<Ø‹øÌfiû+30,ñ€àæDə∑Ó≤]›F◆[wi_MíQcZÌüƒ˙əÎ“ÎòYß¢ÏhØfl(¶<~Ú™zó¸7«ÇÃPl&ÈÇ°Z¥˙émç§¸Ø8–îÛ5ÃB◆–‘á–p&µ¿Ê®!oÛ¢|ŒªÑYU›ΩÈÊªP5NÅ¢~)ª,ë>°¢åÕªœœ5QÜ(»rfò»˘É%/¶€ÔÅÅ Üöu…`
Çl.ïæÈ–)»y£uŒºö#«€ùKÿd˘Ÿ®^rLPΣÄ%ê®M7·}ΔEÄpV‡/:éÚÅ¶jÚÚ·ÆV9Œ       c>©ΔÒ˚–ÇÀŒ†,–h#∏∫¥◊(ß±_«ü‹fiã˜   %éŸ˜πaÄÈìé
Ép±c;!Á˜\änπ5IpØy        ‡»›N?·PÍ{Ô](JΩvX:ëHäZN"ÈÓ–jZE1è/©˘ÃVZUXÕ\ç.cıHE÷qY°‡˜7ò°7#ñ*7Ôãç?c(ÁÍ%¢=ºú)Î!¥∫Ñ{h˙ßíUíbÚDZù@gÓÑn¨°ÒoR I6◊=È§M,6rìd:–†ñµQ∏•Õƒ5†°Sə<çÓÍØèŒê[≈vêÙ«SB†È/¿√<_ÛQ†~ÏÄ◊ƒR˙i®/‡¨{˘:èÃ–oÿ–ü·¸òK§zÁïπY≠ÑRÍœ™–
≈Îø••Äï(fif6má_Øº.Ñ:¥/ëΔgü∏8uÕ˜Ü–åmÌ>~ã˜8
XK§H<>∞tÂ¢¯'%1Ì.{ïJ]4…ïM7‡ÄOc8"hÌJ
Ì3€ãß
ë#~t¸°ç
[R˝¥)ù¨aè|„√ Ö$W≈ÐIzÇÆŸ        j@–ı‘záXÕEY6Ú∫o0Æ¥ãŒ)¨z’“ªª/r€ìì”˜5f√«W÷Á$M;–ƒ±Ö“q
h%Ø∂û÷‹y–B˜fé Â>œÔ÷ÔQ€+¸s˙ ZKÛÆflJ§)PÙÑYHfiÇü·Ùñ
†l◊pa&2pr¢äß–Ç´”
ÔÊAflämêÌ÷¯
ÑíDπ3Ô3çZ»\^9Õæ≈ù¶†ü}vf!ÏÑwÎə∂e§$ìΩ«<ÕÔè∞;úAã˜/”^∫I–ÚÅ8wfiA/Ê®˝nÛH2¶‘#~£Ï_i{·åÎa“AÃíç™Ù24–∏a\∑2^%#B±Ô1nNù6j0â›◆˝Ôå ™πÕ(uÀƒ6≥y;ç‘/HcÀÙµòsë†/4Å‡Ï™Ä§≥âô<≤
 –él›î™{RM@ƒg
'À©‰
"Ω(%à)l$f*[´fl}º¨o‘Sm[óÁÚµZB≠·¥‡®·fiê«™™H¯3Ej_æTµSÑxní
ò~e/x";sFçrR®Î=≤l¯@:qY†◊ämr|kå◊64∏ãä=û`bT,˙0Ô˙•`÷0kî!ªˆn<fiÚ®≥7J–éIÃ∫U»˘«G»0eùΔLí§–â2[¬≤LÔï™mN◊É, l≤|–b/Õ?¯ò©P±”TqLî≠Cª#id2Í∑síÿœ
W/Ñ!œrñÀÚKc–lÇYVj‘ÉÎ¬Â\Âàû√Ÿ/êuÜ¯√:ÿWç¢rë3˜;ºêû9–TP˜˜·@q¯xÑYÅ∏0îßöç$¨&W(‹,2ı™:)ü’ï°9À:iiv< Äo∏íwHo„œûÔ.Ÿ˜>√∫˜j °å::“JOΩJ«áÿ¸“4V
¢·¨flœâiÇ˜Rãò3√Æ≤\0p
CéœHVrÊÉlN'Ÿõfiø j^√M1åû≤æÑ
"åhºqå‹Ë™ÑF≠Œ >Îí2xÛ˜°9;.æöÏÓ ™™
]ÑC}†x∞}£Ôö~˜^@ÃpQÅè•£º•˘˙       éì–†‹5{”∏≠k2ÜëΩQaŒe≥∏‘˜Ôµ*êÙ,ÎS™ã,vY0˚Δ^R∫˜–`nMLÙzπÅ«øN>◆∫3b«Œƒ$Å*WDR/C≥≠°ÓÆ∫∏Δ$§        [ÓK≠Á1ÃÂ,ár·&c¥|Ì*fi‰√–I˜A„ùW¥
gÒØXñè,9€\h¸aÏ◊sPÅ˜ŒL≥9£˙sÃ·˘Ê„ûõÅ õ™ty£ÕI–˜˘ †M˜nMonS≤¢ù!Á,°ÿES^xR\l'¥∫πó>H¢C∑«Ö”1fiµa
„íP)O3ÊÛÎ¶Vï‰+≤uæqM‡Ò0XX¸oJ”R¬}î´©˘ÕYö+ñ\πb∏söÊs,_'˘Bd)Æƒ∫Lgπ¬
ùB◆¢oÚze\;û
HQnq&ôÏZÛÛ”!caö∑∫+·{≠Í?ÔJi‰%5∏ρéΩ€õq„ûˆ ¯/≠!Q¸ËTÙ.L¸/Aó«›˘ºî<NR¬˜OÅSÿ¸Ã√úEÄ4•YÙ8π^•™ä≥Ó™|<©M◊ÓÅoÌ/»ã˘4◆ò{Δn≤Ûxm̃Ő@,/D¸r$æ≠‡M
ƒWÄ9–ª
S(S˜^S/G!·&iÄ{¯ÖRC4˜x›fuwUE¸≈(°7¸ÖM_›uÇÇ¬˜/h     ÌòZ'zß„§.∫7Yo    ë¯mv¯Ô!∫®Vñ˘B≤'…Ü0%&£p∞ç©™I!ñ)Hª≈ÉàQp”}$SæKÅ®RL‘LX8Tv+#PX¸flS˚2µ0ëlI∏QkFpÌ8ØE8¸Èœ•Ú«"ñA…‘Ëó7pfquÜœÄ5°§SÅÈè„ə¸±M/1„µú™>∑KÃó9V3˝ŸéΔ…
Ù       Δ≠
5s∑≤Ëz≈2√6mq◆¸°πZOÅãP¸≥èÉóBÌ˘7"&Ï¬ø*$_ôíflp$‘√Ï∑ij8 izá;D˜T˚I\Ü@∏nª7Ut∑'əÅÕ™ê®ΩKé“248à}Mg≥íå@ù13Hz¯0¸T◊“.6Âh‘kÑjå}.–,/º˘,∑Aã/QÌ7TÁ»p0≥·k◆Â‹kÜüÒwBEzi◆‡˙$]Ω=ã'UÄµΔ“[fiÜV∑VD¸ `i#É HkÆ
e…!     =Ê¯®Ú     ÎÈáÿE'>øÑ~˘¢*3ä‰ÔÊDíJ–Y≠ÖC@©AŸ_<7^0@‹Hv˘Çfiv0]Oə{∫ƒfl◊¢Ĭ>g>g<g|)əm6Á£.ÇåÿΔ…≠ëò™
.ïÁé*5œ";d(ÃÆ@.•~ã˘I¶@◊◆$Ú$pª]Ñ†vµ˙êàP(Í.√H;∫û`yCé1áÜç|50ªE®££ôµìüàr2÷‹Æç˜a≥ıdue!'%ïπ¥ÓÙ˙äñ‘Q%Œƒ/}A,O„ã!'baEıfl(´m≠k∞
∫Ô√=µóí¿≥~™¸.¸xãBΔ}˘Îå∫ƒ|≥áZÕΔ˜#XÈõÄíñ1C◊ç|MãR ¿í©]/«Ôt @ymd÷oŸxüM†®óµµ∫Δ6_dÑQÏœ˘ÀNŒ?œúå’Î–TÚÄ–ì#?ÅÅÉ oÙ,∏~)<–ë|cçñ9ò{ÜßÌ]qü¸KÃwI˜#^L¿AXgF€ø0LT
bQ›<ãÈx^
?qöYlæ´Eò…€ó,~Àì≤íü0–Ô^…épbäüôÕ] 7r+ êoø¿¢Êƒdæòúfi∆c™»õá~yô¢:/õóSå„Á3˘ß√ÕD
Pí4˘∑ù'/©÷˙
ãÛÛje÷pòé˘◆‹?≠fiìÛ]2
Ω„Wj€n„oP‘ΩJ*∫Á:c«h§7Ì
e…ÿ˘ ∂æ:'
```

Decryption

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import os

# File paths
input_file = "/Users/nirudeeswar/Desktop/vein_binary_data_encrypted_aes.txt"  # Encrypted file path
output_file = "/Users/nirudeeswar/Desktop/vein_binary_data_decrypted.txt"  # Decrypted file path

# AES key (same as encryption)
key = b'mysecurekey12345'  # 16 bytes key
iv = b'initialvector123'  # 16 bytes IV (Initialization Vector)

# Check if the input file exists
if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Read the encrypted file
with open(input_file, 'rb') as f:
    encrypted_data = f.read()

# Decrypt the data
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)  # Unpad after decryption

# Write the decrypted data to the output file
with open(output_file, 'wb') as f:
    f.write(decrypted_data)

print(f"File decrypted successfully. Saved to {output_file}")
```

```python
from Crypto.Cipher import DES
from Crypto.Util.Padding import unpad
import os

# Update the input file path to the correct location
input_file = "/Users/nirudeeswar/Documents/vein_binary_data_encrypted_des.txt"
output_file = "/Users/nirudeeswar/Documents/vein_binary_data_decrypted_des.txt"

# Debug: Print the input file path
print(f"Trying to open file at: {input_file}")

# DES key (8 bytes). Ensure it is the correct length.
key = b'my8bytekey'  # 8-byte key for DES

# Check if the input file exists
if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Read the encrypted file
with open(input_file, 'rb') as f:
    encrypted_data = f.read()

# Decrypt the data using DES in ECB mode (if it was encrypted this way)
cipher = DES.new(key, DES.MODE_ECB)  # DES in ECB mode (no IV needed)
decrypted_data = unpad(cipher.decrypt(encrypted_data), DES.block_size)  # Unpad

# Write the decrypted data to the output file
with open(output_file, 'wb') as f:
    f.write(decrypted_data)

print(f"File decrypted successfully. Saved to {output_file}")
```

DNA

```python
from Crypto.Cipher import DES, AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import os

# Define file paths (update with your actual username)
username = "nirudeeswar"
input_file_path = f'/Users/{username}/Desktop/dnasequence.txt'
des_encrypted_file_path = f'/Users/{username}/Desktop/des_encrypted.txt'
aes_encrypted_file_path = f'/Users/{username}/Desktop/aes_encrypted.txt'

# Keys for DES and AES
des_key = get_random_bytes(8)  # DES requires an 8-byte key
aes_key = get_random_bytes(16) # AES requires a 16-byte key (128-bit)

# DES Encryption function
def des_encrypt(data, key):
    cipher = DES.new(key, DES.MODE_ECB)
    padded_data = pad(data, DES.block_size)
    encrypted_data = cipher.encrypt(padded_data)
    return encrypted_data

# AES Encryption function
def aes_encrypt(data, key):
    cipher = AES.new(key, AES.MODE_ECB)
    padded_data = pad(data, AES.block_size)
    encrypted_data = cipher.encrypt(padded_data)
    return encrypted_data

# Check if input file exists
if not os.path.exists(input_file_path):
    print(f"File not found: {input_file_path}")
else:
    # Read the DNA sequence from the input file
    with open(input_file_path, 'r') as f:
        dna_sequence = f.read().strip().encode()  # Read, strip whitespace, and encode as bytes

    # Encrypt the DNA sequence using DES
    print("Encrypting with DES...")
    des_encrypted_data = des_encrypt(dna_sequence, des_key)

    # Write DES encrypted data to a text file in hexadecimal format
    with open(des_encrypted_file_path, 'w') as f:
        f.write(des_encrypted_data.hex())
    print(f"DES Encrypted data saved at: {des_encrypted_file_path}")

    # Encrypt the DES-encrypted data using AES
    print("Encrypting DES output with AES...")
    aes_encrypted_data = aes_encrypt(des_encrypted_data, aes_key)

    # Write the AES encrypted data to a text file in hexadecimal format
    with open(aes_encrypted_file_path, 'w') as f:
        f.write(aes_encrypted_data.hex())
    print(f"AES Encrypted data saved at: {aes_encrypted_file_path}")

    # Output the encryption keys
    print(f"\nDES Key (hex): {des_key.hex()}")
    print(f"AES Key (hex): {aes_key.hex()}")
```

```
Encrypting with DES...
DES Encrypted data saved at: /Users/nirudeeswar/Desktop/des_encrypted.txt
Encrypting DES output with AES...
AES Encrypted data saved at: /Users/nirudeeswar/Desktop/aes_encrypted.txt

DES Key (hex): acff85105f722cfd
AES Key (hex): 57174058f30d9363b043d7225d29a4c3
```

All decryption during unlock

```python
from Crypto.Cipher import DES, AES
from Crypto.Util.Padding import pad, unpad
import os

# Function to verify if the file exists
def verify_file(file_path):
    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return False
    return True

# Function to read binary data from a file
def read_binary_file(file_path):
    with open(file_path, 'rb') as file:
        return file.read()

# Function to save data to a file
def save_binary_file(output_file_path, data):
    with open(output_file_path, 'wb') as file:
        file.write(data)
    print(f"Data saved at: {output_file_path}")

# Function to read text from a file
def read_text_file(file_path):
    with open(file_path, 'r') as file:
        return file.read()

# Function to save text to a file
def save_text_file(output_file_path, data):
    with open(output_file_path, 'w') as file:
        file.write(data)
    print(f"Text file saved at: {output_file_path}")

# Function to decrypt AES
def decrypt_aes(encrypted_data, key):
    aes_cipher = AES.new(key, AES.MODE_ECB)
    decrypted_data = unpad(aes_cipher.decrypt(encrypted_data), AES.block_size)
    return decrypted_data

# Function to decrypt DES with added padding check
def decrypt_des(encrypted_data, key):
    # Ensure the data is a multiple of 8 bytes (DES block size)
    if len(encrypted_data) % DES.block_size != 0:
        # If the data size isn't a multiple of 8, apply padding to make it align with DES block size
        padded_data = encrypted_data + b'\0' * (DES.block_size - len(encrypted_data) % DES.block_size)
    else:
        padded_data = encrypted_data

    des_cipher = DES.new(key, DES.MODE_ECB)
    try:
        decrypted_data = unpad(des_cipher.decrypt(padded_data), DES.block_size)
    except ValueError as e:
        print(f"Error during DES unpadding: {e}")
        exit()
    return decrypted_data
```

```python
# Paths and Keys
aes_encrypted_file_path = '/Users/nirudeeswar/Desktop/aes_encrypted_data.aes'
des_encrypted_file_path = '/Users/nirudeeswar/Desktop/des_encrypted_data.des'
output_file_path = '/Users/nirudeeswar/Desktop/decrypted_original.txt'
new_file_path = '/Users/nirudeeswar/Desktop/minutiae_binary_data.txt'  # File to compare

aes_key = b'securekey1234567'  # Must be 16, 24, or 32 bytes
des_key = b'secretk1'  # Must be 8 bytes

# Step 1: Verify if AES-encrypted file exists
if not verify_file(aes_encrypted_file_path):
    print("AES-encrypted file not found. Exiting.")
    exit()

# Step 2: Decrypt AES to obtain DES-encrypted data
aes_encrypted_data = read_binary_file(aes_encrypted_file_path)
try:
    des_encrypted_data = decrypt_aes(aes_encrypted_data, aes_key)
    print("AES decryption successful.")
except ValueError as e:
    print(f"Error during AES decryption: {e}")
    exit()

# Step 3: Save DES-encrypted data for verification (optional)
save_binary_file(des_encrypted_file_path, des_encrypted_data)

# Step 4: Decrypt DES-encrypted data to obtain original text
try:
    original_text = decrypt_des(des_encrypted_data, des_key).decode('utf-8')
    print("DES decryption successful.")
except ValueError as e:
    print(f"Error during DES decryption: {e}")
    exit()

# Step 5: Save the decrypted original text to a file
save_text_file(output_file_path, original_text)

# Step 6: Compare the original text with the new file
if not verify_file(new_file_path):
    print("New file for comparison not found. Exiting.")
    exit()

new_file_text = read_text_file(new_file_path)
if original_text == new_file_text:
    print("Access Granted: Files match.")
else:
    print("Access Denied: Files do not match.")
```

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import os

# File paths
input_file = "/Users/nirudeeswar/Desktop/vein_binary_data_encrypted_aes.txt"  # Encrypted file path
output_file = "/Users/nirudeeswar/Desktop/vein_binary_data_decrypted.txt"  # Decrypted file path

# AES key (same as encryption)
key = b'mysecurekey12345'  # 16 bytes key
iv = b'initialvector123'  # 16 bytes IV (Initialization Vector)

# Check if the input file exists
if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Read the encrypted file
with open(input_file, 'rb') as f:
    encrypted_data = f.read()

# Decrypt the data
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)  # Unpad after decryption

# Write the decrypted data to the output file
with open(output_file, 'wb') as f:
    f.write(decrypted_data)

print(f"File decrypted successfully. Saved to {output_file}")
```

```python
from Crypto.Cipher import DES
from Crypto.Util.Padding import unpad
import os

# Update the input file path to the correct location
input_file = "/Users/nirudeeswar/Documents/vein_binary_data_encrypted_des.txt"
output_file = "/Users/nirudeeswar/Documents/vein_binary_data_decrypted_des.txt"

# Debug: Print the input file path
print(f"Trying to open file at: {input_file}")

# DES key (8 bytes). Ensure it is the correct length.
key = b'my8bytekey'  # 8-byte key for DES

# Check if the input file exists
if not os.path.exists(input_file):
    print(f"Error: File not found at {input_file}. Please check the path.")
    exit()

# Read the encrypted file
with open(input_file, 'rb') as f:
    encrypted_data = f.read()

# Decrypt the data using DES in ECB mode (if it was encrypted this way)
cipher = DES.new(key, DES.MODE_ECB)  # DES in ECB mode (no IV needed)
decrypted_data = unpad(cipher.decrypt(encrypted_data), DES.block_size)  # Unpad

# Write the decrypted data to the output file
with open(output_file, 'wb') as f:
    f.write(decrypted_data)

print(f"File decrypted successfully. Saved to {output_file}")
```

```python
from Crypto.Cipher import DES, AES
from Crypto.Util.Padding import unpad
import os

# Define file paths (update with your actual username)
username = "nirudeeswar"
des_encrypted_file_path = f'/Users/{username}/Desktop/des_encrypted.bin'
aes_encrypted_file_path = f'/Users/{username}/Desktop/aes_encrypted.bin'
output_decrypted_file_path = f'/Users/{username}/Desktop/decrypted_dna_sequence.txt'

# Replace these with the actual keys used during encryption
des_key = bytes.fromhex(input("Enter the DES key (hex format): "))
aes_key = bytes.fromhex(input("Enter the AES key (hex format): "))

# AES Decryption function
def aes_decrypt(data, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_data = cipher.decrypt(data)
    unpadded_data = unpad(decrypted_data, AES.block_size)
    return unpadded_data

# DES Decryption function
def des_decrypt(data, key):
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_data = cipher.decrypt(data)
    unpadded_data = unpad(decrypted_data, DES.block_size)
    return unpadded_data

# Check if AES-encrypted file exists
if not os.path.exists(aes_encrypted_file_path):
    print(f"File not found: {aes_encrypted_file_path}")
else:
    # Read the AES-encrypted file
    with open(aes_encrypted_file_path, 'rb') as f:
        aes_encrypted_data = f.read()

    # Decrypt the AES-encrypted data
    print("Decrypting with AES...")
    des_encrypted_data = aes_decrypt(aes_encrypted_data, aes_key)

    # Decrypt the DES-encrypted data
    print("Decrypting with DES...")
    original_dna_sequence = des_decrypt(des_encrypted_data, des_key)

    # Write the decrypted DNA sequence to a file
    with open(output_decrypted_file_path, 'w') as f:
        f.write(original_dna_sequence.decode())

    print(f"Decryption complete. Original DNA sequence saved at: {output_decrypted_file_path}")
```

Future works

- Developing a working hardware system

References

1. RF Sensor-Based Liveness Detection Scheme With Loop StabilityCompensation Circuit for a Capacitive Fingerprint SystemWOOJUNG KIM, WOOJIN HONG, TAEKMOO KIM,DONGWOON KIM,AND MYUNGHEE LEE

2. Statistical anti-spoofing method for fingerprint recognition Yosep Park · Unsoo Jang · Eui Chul Lee

3. Deep Representations for Iris, Face, and Fingerprint Spoofing Detection David Menotti Member, , Giovani Chiachia , Allan Pinto, Student Member, William Robson Schwartz, Member, Helio Pedrini, Member,, Alexandre Xavier Falca˜o, Member, and Anderson Rocha, Member

4. Fingerprint Spoof Detector generalisation Tarang Chugh*, Student Member, IEEE, and Anil K. Jain, Life Fellow, IEEE

5. Proactive forensic science in biometrics: Novel materials for fingerprint spoofing Michel Saguy ME1 | Joseph Almog PhD2 Christophe Champod PhD1

6. DeFraudNet:End2End Fingerprint Spoof Detection using Patch Level Attention B.V.S Anusha Sayan Banerjee Subhasis Chaudhuri

7. Anti-spoofing method for fingerprint recognition using patch based deep learning machine Diaa M. Uliyan a,⇑, Somayeh Sadeghi b, Hamid A. Jalab

8. CONTOURLET-BASED FINGERPRINT ANTISPOOFING Shankar Bhausaheb Nikam1 and Suneeta Agarwal2

9. Fingerprint spoofing attacks and their deep learning enabled remediation Ruthbaa Ishfaq, Arvind Selwal, Deepika Sharma