INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Tecnologias Biométricas –

Implementação de um Sistema Biométrico

Luís Miguel Berenguer, 23695

Pedro Pita, 19933

Tomás Ferreira, 19934

2024

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Tecnologias Biométricas – Implementação de um Sistema Biométrico

Elaborado por:

Luís Miguel Berenguer, 23695

Pedro Pita, 19933

Tomás Ferreira, 19934

Orientado por:

Pedro Moreira

Relatório de projeto de Tecnologias Biométricas apresentado na

Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja

2024

Índice

Lista de Figuras

1. Introdução

No âmbito da disciplina de Tecnologias Biométricas, foi nos proposto a realização de um projeto focado na "Implementação de um Sistema Biométrico". Este projeto visa explorar aplicações práticas de tecnologias biométricas, abordando aspetos cruciais como eficácia, eficiência, segurança e privacidade.

Com isso em vista, decidimos, a partir do código desenvolvido em aula, focarmo-nos na impressão digital analisando e melhorando a eficiência será analisada em termos de tempo de resposta e utilização de recursos do sistema, para além de verificarmos a sua eficácia quanto à sua capacidade de identificar e autenticar indivíduos. A segurança será reforçada com medidas para proteger os dados biométricos, através da implementação de criptografia, e a privacidade será garantida através da adoção de práticas específicas.

2. Implementação de um Sistema Biométrico

Para este projeto, concentrámo-nos na análise e otimização do código referente à tecnologia biométrica de impressão digital, desenvolvido durante as aulas.

Nosso objetivo principal foi aprimorar a eficácia do sistema, assegurando uma identificação precisa e eficiente dos utilizadores. A revisão do código incluiu também medidas para reforçar a segurança dos dados biométricos, bem como a implementação de práticas que respeitam a privacidade dos indivíduos envolvidos.

2.1. Explicação Procedimento Experimental

Para conseguirmos obter uma evolução notória deste projeto, realizamos procedimento experimental por fases, em que desenvolvemos exemplos com cada implementação de melhoria do trabalho, a fim de podermos fazer comparações. O procedimento passou da seguinte forma:

1- Cifragem das imagens Biométricas
2- Otimização da performance na comparação das impressões digitais
3- Implementação de sistema de base de dados, e na identificação de utilizadores

O seguinte código representa o esquema da base de dados criada, que providencia uma tabela para armazenar os utilizadores do sistema, uma tabela que se liga com os utilizadores, que armazena todas as impressões digitais de cada utilizador, e por fim uma tabela que guarda os logs do sistema para garantir maior segurança e privacidade nas operações realizadas na BD.

```
conn = sqlite3.connect('localdatabase.db')
cursor = conn.cursor()

# Create the 'users' table if it does not exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        username TEXT UNIQUE NOT NULL,
        name TEXT NOT NULL,
        age INTEGER NOT NULL,
        failed_attempts INTEGER NOT NULL DEFAULT 0,
        bloqued_until DATETIME
    )
''')

# Create the 'fingerprint' table if it does not exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS fingerprint (
        id INTEGER PRIMARY KEY,
        fingerprint_data BLOB NOT NULL,
        user_id INTEGER,
        FOREIGN KEY (user_id) REFERENCES users(id)
    )
''')

# Create the 'command_history' table if it does not exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS command_history (
        id INTEGER PRIMARY KEY,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
        command_text TEXT NOT NULL
    )
''')
```

Figura 2-1 - Estrutura BD

4- Implementação de sistema de autenticação segura de utilizadores

Desenvolvemos um sistema de autenticação seguro, em que o utilizador necessita de inserir a sua sigla e a sua impressão digital, permitindo apenas o acesso às informações biométricas apenas ao utilizador, e caso a mesma esteja incorreta é incrementado o número de tentativas incorretas para aquele utilizador, e passado o limite de tentativas possíveis, o utilizador fica bloqueado durante (x tentativas / limite) minutos, impedidas tentativas de brute force, e só ao acesso da impressão digital do utilizador em específico.

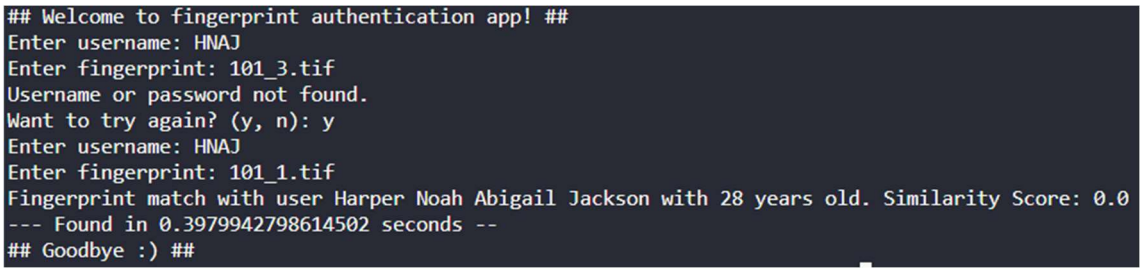

Figura 2-2 - Exemplo execução sistema autenticação

2.2. Segurança

Como foco inicial, procedemos com a implementação de segurança nos dados biométricos para não puderem ser acedidos por utilizadores externos, com isso em vista, decidimos aplicar criptografia nos dados biométricos dos utilizadores, através da cifra de Fernet nas imagens das impressões digitais dos utilizadores. Esta escolha estratégica visa garantir a integridade e confidencialidade dos dados biométricos, assegurando uma camada adicional de proteção.

A cifra de Fernet, conhecida pela sua robustez e eficiência, revela-se uma opção ideal para criptografar as imagens das impressões digitais. Ao utilizar um algoritmo simétrico de chave secreta, proporciona uma segurança sólida, garantindo que apenas os utilizadores autorizados possam aceder e interpretar os dados biométricos. Esta medida não apenas impede acessos não autorizados, mas também eleva o nível de resistência do sistema a possíveis tentativas de ataques externos.

Criamos um módulo de cifragem e decifragem dos ficheiros, em que é só necessário o caminho do ficheiro (cifrado/decifrado) e a chave da cifra. A criação de módulos permite que as funções possam ser utilizadas em todos os ficheiros que as necessitassem.

```python
def encrypt_file(file_path, key):
    # Read the contents of the file
    with open(file_path, 'rb') as file:
        file_data = file.read()

    # Create a Fernet cipher suite using the provided key
    cipher_suite = Fernet(key)

    # Encrypt the file data
    encrypted_data = cipher_suite.encrypt(file_data)

    # Write the encrypted data to a new file with the .enc extension
    #with open(file_path + '.enc', 'wb') as file:
    #    file.write(encrypted_data)

    return encrypted_data
```

Figura 2-3 - encryption_module: Cifragem de ficheiro

```python
def decrypt_file(file_path, key):
    try:
        # Read the encrypted data from the file
        with open(file_path, 'rb') as file:
            encrypted_data = file.read()

        # Create a Fernet cipher suite using the provided key
        cipher_suite = Fernet(key)

        # Decrypt the data
        decrypted_data = cipher_suite.decrypt(encrypted_data)

        # Return the decrypted data
        return decrypted_data
    except InvalidToken:
        #print("Error: Invalid key. Decryption failed.")
        return None
```

Figura 2-4 - encryption_module: Decifragem de ficheiro

Na primeira parte do projeto, usamos um sistema de armazenamento em ficheiros, em que a partir da pasta com as imagens das impressões digitais, gerávamos uma pasta com as impressões digitais cifradas.

```python
def process_files(directory_path, key):
    # Encrypt and decrypt files in the specified directory
    for root, dirs, files in os.walk(directory_path):
        for file in files:
            if file.endswith('.tif'):
                input_file_path = os.path.join(root, file)
                output_file_path = os.path.join('fingerprint_dataset_encrypt', file)

                # Encrypt the file and save it with the .enc extension
                encrypt_file(input_file_path, key.encode())

                try:
                    # Rename the encrypted file to the output directory
                    os.rename(input_file_path + '.enc', output_file_path + '.enc')
                except Exception as e:
                    print(f"Error: {e}")
                    os.remove(input_file_path + '.enc')
```

Figura 2-5 - Cifragem da pasta dos ficheiros de impressões digitais

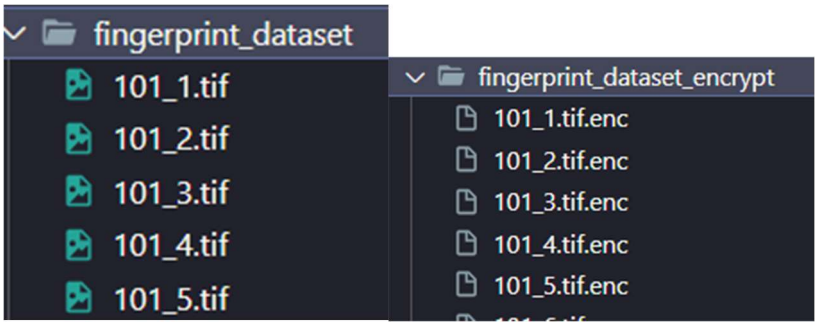

Figura 2-6 - Resultado cifragem das imagens das impressões digitais

Já na segunta fase, para armazenamento das impressções digitais na base de dados, guarda-mos os dados cifrados, e criamos os utilizaadores, de forma randomizada, por cada impressão digital, sendo que poderiamos ter que um utilizador pode ter mais que uma impressão digital pela forma como o sistema foi desenvolvido.

```python
def process_files(directory_path, key):
    # Connect to the database
    cursor = connect_database()

    # Encrypt and decrypt files in the specified directory
    for root, dirs, files in os.walk(directory_path):
        for file in files:
            if file.endswith('.tif'):
                input_file_path = os.path.join(root, file)
                #output_file_path = os.path.join('fingerprint_dataset_encrypt', file)

                # Encrypt the file and save it with the .enc extension
                encrypted_data = encrypt_file(input_file_path, key.encode())

                try:
                    # Create a new user
                    insert_user_command = 'INSERT INTO users (username, name, age) VALUES (?, ?, ?)'

                    user_name = generate_random_name()
                    username = "".join([n[0] for n in user_name.split(' ')])

                    # Generate a random user name and age
                    user_params = (username, user_name, random.randint(10, 80))
                    execute_command_and_log(cursor, insert_user_command, user_params)

                    # Get the ID of the newly inserted user
                    user_id = cursor.lastrowid

                    # Insert a fingerprint for the user
                    insert_fingerprint_command = 'INSERT INTO fingerprint (fingerprint_data, user_id) VALUES (?, ?)'
                    fingerprint_params = (encrypted_data, user_id)
                    execute_command_and_log(cursor, insert_fingerprint_command, fingerprint_params)

                    print(f"User '{user_name}', with username '{username}' with ID {user_id} created successfully with a fingerprint.")
                except Exception as e:
                    print(f"Error: {e}")
```

Figura 2-7 - Criação de utilizadores e cifragem da impressão digital

Além da cifragem dos ficheiros, ao decifrar os ficheiros para serem analisados, utilizamos um sistema de ficheiros temporários seguros, para que mesmo que os ficheiros das impressões digitais decifrados sejam intercetados durante a execução do programa, estes não possam ser lidos.

```python
# Use tempfile to create a secure temporary file
with tempfile.NamedTemporaryFile(delete=False) as temp_file:
    temp_file.write(decrypted_data)
```

Figura 2-8 - Criação de ficheiros temporários seguros

```python
# Delete the temp file decrypted
os.remove(temp_file_path)
```

Figura 2-9 - Eliminação do ficheiro temporário

Também desenvolvemos um sistema de autenticação seguro, em que o utilizador necessita de inserir a sua sigla e a sua impressão digital, e caso a mesma esteja incorreta é incrementado o número de tentativas incorretas para aquele utilizador, e passado o limite de tentativas possíveis, o utilizador fica bloqueado durante (x tentativas / limite) minutos, impedidas tentativas de brute force, e só ao acesso da impressão digital do utilizador em específico.

```python
if authentifing == True:
    failed_attempts += 1

    execute_command_and_log(cursor, "UPDATE users SET failed_attempts = ? WHERE id = ?", (str(failed_attempts), str(selected_user_id),))

    # Check if passes the limit of attempts
    if failed_attempts >= failed_attempts_limit:
        now = datetime.now()

        # Add x minutes to the bloqued time
        new_bloqued_until = now + timedelta(minutes=(failed_attempts / failed_attempts_limit))

        execute_command_and_log(cursor, "UPDATE users SET bloqued_until = ? WHERE id = ?", (str(new_bloqued_until), str(selected_user_id)))

    print(f"Username or password not found.")
    awnser = input("Want to try again? (y, n): ")
    if awnser == 'n':
        authentifing = False

    continue
```

Figura 2-10 - Incrementa o número de tentativas falhadas e o tempo bloqueado

```python
while authentifing:
    username = input("Enter username: ")

    # Get user
    execute_command_and_log(cursor_selected_user, "SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?", (str(username),))
    user_db = cursor_selected_user.fetchone()

    # Define the path to the reference fingerprint image
    #reference_image_path = "101_2.tif"
    reference_image_path = input("Enter fingerprint: ")

    if not user_db:
        print(f"Username or password not found.")
        awnser = input("Want to try again? (y, n): ")
        if awnser == 'n':
            authentifing = False

        continue

    selected_user_id, name, age, failed_attempts, bloqued_until = user_db

    if bloqued_until:
        print(bloqued_until)
        bloqued_until_date = datetime.strptime(bloqued_until, "%Y-%m-%d %H:%M:%S.%f")
        now = datetime.now()
        if bloqued_until_date >= now:
            # Calculate the time difference in seconds
            time_difference_seconds = (bloqued_until_date - now).total_seconds()

            # Calculate hours, minutes, and remaining seconds
            hours, remainder = divmod(time_difference_seconds, 3600)
            minutes, seconds = divmod(remainder, 60)

            print(f"Too many attempts, user blocked for {int(hours)} hours, {int(minutes)} minutes, and {int(seconds)} seconds.")

            awnser = input("Want to try again? (y, n): ")
            if awnser == 'n':
                authentifing = False

            continue
```

Figura 2-11 - Verifica se o utilizador existe ou está bloqueado

```
## Welcome to fingerprint authentication app! ##
Enter username: HNAJ
Enter fingerprint: 101_2.tif
2024-01-05 02:54:35.805634
Too many attempts, user blocked for 0 hours, 1 minutes, and 19 seconds.
```

Figura 2-12 - Exemplo utilizador bloqueado

Por fim, o nosso sistema de base de dados oferece um sistema de logs dos comandos que nos permite verificar todas as operações que foram realizadas na BD, podendo assim visualizar que tipo de acessos foram realizados.

```python
def execute_command_and_log(cursor, command, params=None):
    """
    Execute the given SQL command and store it in the command_history table.
    """
    global conn

    if not cursor:
        cursor = create_db_cursor()

    # Log the command in command_history
    timestamp = datetime.now()
    cursor.execute('INSERT INTO command_history (timestamp, command_text) VALUES (?, ?)', (timestamp, command))

    # Execute the command with optional parameters
    if params is not None:
        cursor.execute(command, params)
    else:
        cursor.execute(command)

    # Commit the changes
    conn.commit()
```

Figura 2-13 - Execução de comandos BD e armazenamento de log

```python
def get_database_logs(order="DESC"):
    global cursor
    execute_command_and_log(cursor, f'SELECT * FROM command_history ORDER BY id {order}')
    logs = cursor.fetchall()

    for log in logs:
        id, timestamp, command_text = log
        print(f"Time: {timestamp}; Query: {command_text}")
```

Figura 2-14 - Obtenção de todos os logs

```
PS D:\Documents\Universidade\Mestrado\TB\Trabalho Prático v2\db_example> python3 .\database_logs.py
Time: 2024-01-06 19:09:42.492774; Query: SELECT * FROM command_history ORDER BY id DESC
Time: 2024-01-06 19:01:17.516306; Query: UPDATE users SET failed_attempts = ?, bloqued_until = ? WHERE id = ?
Time: 2024-01-06 19:01:17.118311; Query: SELECT id, fingerprint_data, user_id FROM fingerprint WHERE user_id = ? ORDER BY id
Time: 2024-01-06 19:01:14.678975; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
Time: 2024-01-06 19:01:10.528444; Query: UPDATE users SET failed_attempts = ? WHERE id = ?
Time: 2024-01-06 19:01:10.074654; Query: SELECT id, fingerprint_data, user_id FROM fingerprint WHERE user_id = ? ORDER BY id
Time: 2024-01-06 19:01:07.815002; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
Time: 2024-01-06 18:59:43.271776; Query: UPDATE users SET failed_attempts = ?, bloqued_until = ? WHERE id = ?
Time: 2024-01-06 18:59:42.878811; Query: SELECT id, fingerprint_data, user_id FROM fingerprint WHERE user_id = ? ORDER BY id
Time: 2024-01-06 18:59:36.564305; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
Time: 2024-01-06 18:59:28.763657; Query: UPDATE users SET failed_attempts = ? WHERE id = ?
Time: 2024-01-06 18:59:27.242807; Query: SELECT id, fingerprint_data, user_id FROM fingerprint WHERE user_id = ? ORDER BY id
Time: 2024-01-06 18:59:19.695075; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
Time: 2024-01-05 02:57:20.113906; Query: UPDATE users SET failed_attempts = ?, bloqued_until = ? WHERE id = ?
Time: 2024-01-05 02:57:19.756326; Query: SELECT id, fingerprint_data, user_id FROM fingerprint WHERE user_id = ? ORDER BY id
Time: 2024-01-05 02:57:15.477510; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
Time: 2024-01-05 02:55:16.220967; Query: SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?
```

Figura 2-15 - Logs BD

2.3. Eficiência

Como segunda fase, procedemos com a análise do tempo de resposta do sistema e o consumo de recursos. O computador usado para testes, por ter cerca de 7 anos e ser mais antigo, conseguimos perceber a lentidão que um sistema não otimizado pode levar a processar as imagens, sendo o primeiro sistema desenvolvido o seguinte:

```
101_2.tif.enc: Fingerprints match! Similarity Score: 0.0
--- Found in 117.95053720474243 seconds --
-- CPU Usage at 2024-01-06 17:57:23.725769: 8.6%
-- Memory Usage at 2024-01-06 17:57:23.729777:
Total: 16268.921875 MB
Used: 9679.52734375 MB
Usage Percentage: 59.5%
-- Disk Usage at 2024-01-06 17:57:23.730776:
Total: 237.87206649780273 GB
Used: 135.56541442871094 GB
Free: 102.3066520690918 GB
Usage Percentage: 57.0%
```

Figura 2-16 - Reconhecimento Impressão digital (Não otimizado)

Com este print, conseguimos perceber que só para comparar uma impressão digital leva 118 segundos e consome imensos recursos do sistema. Após uma análise do código, percebemos que o problema se encontrava na extração de minúcias e na comparação das duas impressões digitais.

```python
def extract_minutiae(binary_image):
    # Find contours in the binary image
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    minutiae = []
    for contour in contours:
        # Filter small contours
        if cv2.contourArea(contour) > 50:
            for point in contour:
                x, y = point[0]
                minutiae.append((x, y))

    return minutiae
```

Figura 2-17 - Extração de minúcias de Impressões digitais (Original)

```python
def euclidean_distance(point1, point2):
    return np.linalg.norm(np.array(point1) - np.array(point2))

def match_fingerprints(minutiae1, minutiae2):
    # Calculate distances between minutiae points using Euclidean distance
    distances = np.zeros((len(minutiae1), len(minutiae2)))
    for i, m1 in enumerate(minutiae1):
        for j, m2 in enumerate(minutiae2):
            #distances[i, j] =  np.linalg.norm(np.array(m1[:2]) - np.array(m2[:2]))
            distances[i, j] = euclidean_distance(m1, m2)

    # Calculate similarity score
    min_distances = np.min(distances, axis=1)
    similarity_score = np.mean(min_distances)

    return similarity_score
```

Figura 2-18 - Comparação Impressões digitais (Original)

Percebemos que poderíamos aproveitar a biblioteca do numpy para nos fornecer cálculos mais eficientes.

```python
def extract_minutiae(binary_image):
    # Find contours in the binary image
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    minutiae = []
    for contour in contours:
        # Filter small contours
        if cv2.contourArea(contour) > 50:
            # Append minutiae points from the contour to the list
            minutiae.extend(contour[:, 0, :].tolist())

    return minutiae
```

Figura 2-19 - Extração de minúcias de Impressões digitais (Otimizado)

```python
def match_fingerprints(minutiae1, minutiae2):
    """
    (Optimized euclidean distance) Calculate distances between minutiae points using Euclidean distance:
    - np.array(minutiae1)[:, None, :2] - This converts the list of minutiae points in minutiae1 to a NumPy ar
    - np.array(minutiae2)[None, :, :2] - Similar to the first part, this converts the list of minutiae points
    - Subtracting the two arrays - The code then subtracts the two arrays obtained in the first and second st
    np.linalg.norm(..., axis=-1):
    - Finally, the np.linalg.norm function calculates the Euclidean norm (distance) along the last axis (axis
    """
    distances = np.linalg.norm(np.array(minutiae1)[:, None, :2] - np.array(minutiae2)[None, :, :2], axis=-1)

    # Calculate similarity score
    min_distances = np.min(distances, axis=1)
    similarity_score = np.mean(min_distances)

    return similarity_score
```

Figura 2-20 - Comparação Impressões digitais (Otimizado)

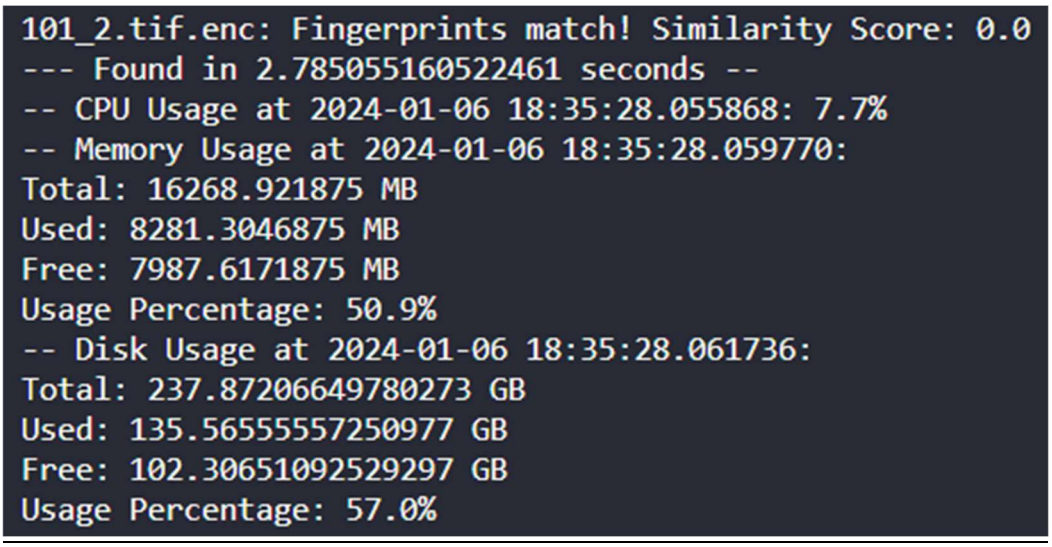

Figura 2-21 - Reconhecimento Impressão digital (Otimizado)

Após a otimização realizada, notamos uma melhoria significativa de performance, sendo que levou apenas 3 segundos, para encontrar a mesma impressão digital, para além de uma diminuição do uso de recursos, como o CPU e a RAM.

2.4. Eficácia

A distância de Euclides [3] é frequentemente utilizada para comparar eficazmente duas impressões digitais no contexto de sistemas biométricos. Essa métrica de distância é aplicada para avaliar a similaridade ou dissimilaridade entre duas características biométricas, como as representadas pelas imagens das impressões digitais.

Ao utilizar a distância de Euclides na comparação de impressões digitais, os vetores que representam as características biométricas são tratados como pontos no espaço tridimensional (ou multidimensional, dependendo da dimensionalidade dos dados). A distância euclidiana entre esses pontos é então calculada através da fórmula matemática:

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Figura 2-22 - Distância de Euclides

Quando aplicada a impressões digitais, os pontos no espaço representam as características específicas das impressões, como minúcias ou pontos singulares. Quanto menor a distância euclidiana entre os pontos, maior é a similaridade entre as impressões digitais. Caso a distância seja pequena o suficiente, as impressões podem ser consideradas como provenientes da mesma fonte, no caso o threshold mais seguro que analisamos foi 4, que garante uma eficácia maior na validação das impressões digitais.

```
# Set a similarity threshold
threshold = 4
```

Figura 2-23 - Definição do threshold

```
def match_fingerprints(minutiae1, minutiae2):
    """
    (Optimized euclidean distance) Calculate distances between minutiae points using Euclidean distance:
    - np.array(minutiae1)[:, None, :2] - This converts the list of minutiae points in minutiae1 to a NumPy array and adds a new axis (None or np.newaxis) a
    - np.array(minutiae2)[None, :, :2] - Similar to the first part, this converts the list of minutiae points in minutiae2 to a NumPy array. The [None, :,
    - Subtracting the two arrays - The code then subtracts the two arrays obtained in the first and second steps. This subtraction is element-wise, meaning
    np.linalg.norm(..., axis=-1):
    - Finally, the np.linalg.norm function calculates the Euclidean norm (distance) along the last axis (axis=-1). This results in an array of Euclidean di
    """
    distances = np.linalg.norm(np.array(minutiae1)[:, None, :2] - np.array(minutiae2)[None, :, :2], axis=-1)

    # Calculate similarity score
    min_distances = np.min(distances, axis=1)
    similarity_score = np.mean(min_distances)

    return similarity_score
```

Figura 2-24 - Comparação das impressões digitais com a distância de euclides

```
# Match fingerprints and calculate similarity score
match_score = match_fingerprints(minutiae_reference, minutiae_current)

# Compare the similarity score with the threshold
if match_score < threshold:
  print(f"{file_enc}: Fingerprints match! Similarity Score: {match_score}")
  print("--- Found in %s seconds --" % (time.time() - start_time))
else:
  print(f"{file_enc}: Fingerprints do not match! Similarity Score: {match_score}")
```

Figura 2-25 - Validação da comparação das impressões digitais

Relativamente ao sistema de autenticação, o sistema será mais eficaz pelo facto de que para além da impressão digital, o sistema terá que também identificar o utilizador que pretende aceder.

```python
username = input("Enter username: ")

# Get user
execute_command_and_log(cursor_selected_user, "SELECT id, name, age, failed_attempts, bloqued_until FROM users WHERE username = ?", (str(username),))
user_db = cursor_selected_user.fetchone()

# Define the path to the reference fingerprint image
#reference_image_path = "101_2.tif"
reference_image_path = input("Enter fingerprint: ")

if not user_db:
    print(f"Username or password not found.")
    awnser = input("Want to try again? (y, n): ")
    if awnser == 'n':
        authentifing = False

    continue

selected_user_id, name, age, failed_attempts, bloqued_until = user_db

if bloqued_until:
    #print(bloqued_until)
    bloqued_until_date = datetime.strptime(bloqued_until, "%Y-%m-%d %H:%M:%S.%f")
    now = datetime.now()
    if bloqued_until_date >= now:
        # Calculate the time difference in seconds
        time_difference_seconds = (bloqued_until_date - now).total_seconds()

        # Calculate hours, minutes, and remaining seconds
        hours, remainder = divmod(time_difference_seconds, 3600)
        minutes, seconds = divmod(remainder, 60)

        print(f"Too many attempts, user blocked for {int(hours)} hours, {int(minutes)} minutes, and {int(seconds)} seconds.")

        awnser = input("Want to try again? (y, n): ")
        if awnser == 'n':
            authentifing = False

        continue
```

Figura 2-26 - Eficácia sistema autenticação

```
## Welcome to fingerprint authentication app! ##
Enter username: HNAJ
Enter fingerprint: 101_3.tif
Username or password not found.
Enter username: HNAJ
Enter fingerprint: 101_1.tif
Fingerprint match with user Harper Noah Abigail Jackson with 28 years old. Similarity Score: 0.0
--- Found in 0.3979942798614502 seconds --
## Goodbye :) ##
```

Figura 2-27 - Execução sistema autenticação

2.5. Privacidade

A fim de garantir a privacidade do sistema, utilizamos cifragem dos dados com a cifra de Fernet [4], o que permite que cada utilizador tenha a sua chave e só esse utilizador pode aceder à sua biometria, isto poderia ser implementado se o utilizador tivesse de colocar a sigla e a sua password, e depois para se autenticar teria que adicionar a sua impressão digital também.

Para este projeto, só identificamos com uma chave de Fernet do sistema, porém garantimos a privacidade já que o utilizador precisa da sigla correta para aceder à sua biometria e compará-la para se autenticar, além de sistema de logs e proteção contra ataques brute force, como abordado no Ponto 2.2.

3. Conclusão

Em conclusão, a aplicação da distância de Euclides na comparação de impressões digitais representa uma abordagem eficaz e amplamente utilizada em sistemas biométricos. Ao tratar as características biométricas como pontos no espaço, essa métrica possibilita a avaliação da similaridade entre duas impressões digitais de forma quantitativa. A utilização dessa medida de distância contribui para a eficácia na identificação e autenticação de indivíduos, permitindo que sistemas biométricos determinem a proximidade ou distância entre as características específicas das impressões digitais.

No entanto, é crucial reconhecer que a escolha da métrica de distância, seja ela de Euclides ou outra, deve ser feita considerando a natureza e as características específicas do sistema em questão. Além disso, a implementação eficaz de medidas de segurança, como a cifra de Fernet mencionada anteriormente, é vital para garantir a proteção adequada dos dados biométricos. A combinação cuidadosa desses elementos contribui para o desenvolvimento de sistemas biométricos robustos, seguros e capazes de atender aos requisitos rigorosos de identificação e autenticação.

Bibliografia

[1] Moodle – Tecnologia Biométricas, 2023, Pedro Moreira, https://cms.ipbeja.pt/course/view.php?id=2062.

[2] Python, https://www.python.org.

[3] Euclidean Distance | Calculation, Formula & Examples, 2023, Study.com, https://study.com/academy/lesson/euclidean-distance-calculation-formula-examples.html.

[4] What is Fernet? Secure data encryption in Python, 2023, JOSH LAKE, https://www.comparitech.com/blog/information-security/what-is-fernet/.