BÀI 4: SOCKET, WEBSOCKET, TRAO ĐỔI KHOÁ VÀ HÀM BĂM

Trong bài thực hành này, chúng ta sẽ tìm hiểu về Socket và WebSocket, cũng như cách triển khai chúng trong Python bằng module Python Socket và framework Tornado. Ngoài ra, chúng ta sẽ khám phá phương pháp Trao đổi khóa bằng thuật toán Diffie-Hellman và các hàm băm như MD5, SHA-256, SHA-3, và Blake2. Bài học sẽ đi sâu vào cách thức hoạt động, đặc điểm và ứng dụng của từng thuật toán trong việc đảm bảo tính toàn vẹn của dữ liệu.

4.1 SOCKET VÀ MODULE PYTHON SOCKET

Socket (hay còn gọi là ổ cắm) là một giao diện chuẩn giúp các ứng dụng trên các thiết bị khác nhau trong mạng có thể giao tiếp và trao đổi dữ liệu. Socket có vai trò quan trọng trong việc thiết lập kết nối và truyền dữ liệu giữa các thiết bị trong mạng máy tính, làm cho quá trình trao đổi thông tin giữa các ứng dụng trở nên linh hoạt. Nhờ vậy, việc giao tiếp có thể diễn ra trên nhiều loại thiết bị, chẳng hạn như máy tính cá nhân, máy chủ, điện thoại di động và các thiết bị thuộc hệ sinh thái IoT (Internet of Things).

Socket được xác định bởi một địa chỉ IP và một số cổng. Địa chỉ IP giúp nhận diện máy tính hoặc thiết bị trong mạng, còn số cổng xác định ứng dụng cụ thể đang hoạt động trên thiết bị đó. Khi một ứng dụng cần kết nối với một máy chủ hoặc thiết bị khác, nó sẽ sử dụng Socket để gửi yêu cầu kết nối qua mạng. Có hai loại chính của Socket:

- **TCP Socket:** Được sử dụng trong giao thức TCP để thiết lập kết nối đáng tin cậy giữa các thiết bị.
- **UDP Socket:** Sử dụng giao thức UDP, là một giao thức giao tiếp không đảm bảo tính đáng tin cậy cao như TCP, nhưng nó có tốc độ truyền dữ liệu nhanh hơn và ít tốn kém hơn về tài nguyên mạng.

Trong Python, module "**socket**" cung cấp giao diện cho việc làm việc với các Socket, cho phép ứng dụng Python tạo, kết nối và tương tác với các Socket TCP/IP và UDP. Module "**socket**" trong Python cung cấp một cách mạnh mẽ và linh hoạt để thực hiện các hoạt động mạng cơ bản và là một công cụ quan trọng cho việc phát triển ứng dụng liên quan đến mạng.

4.2 WEBSOCKET VÀ TORNADO FRAMEWORK

WebSocket là một công nghệ truyền thông cho phép thiết lập một kết nối liên tục và hai chiều giữa trình duyệt và máy chủ thông qua một kênh giao tiếp duy nhất. Công nghệ này giúp truyền tải dữ liệu theo thời gian thực giữa các thiết bị khác nhau một cách hiệu quả, giảm độ trễ so với các phương thức truyền thống. Điểm khác biệt chính giữa WebSocket và các phương pháp như HTTP là khả năng duy trì kết nối luôn mở.

Tornado là một framework web mạnh mẽ viết bằng Python, ban đầu được phát triển bởi công ty FriendFeed (sau đó được Facebook mua lại). Framework này được thiết kế để hỗ trợ các ứng dụng web thời gian thực, với khả năng quản lý hàng nghìn kết nối đồng thời một cách hiệu quả. Tornado thường được dùng để xây dựng các ứng dụng cần thời gian thực như dịch vụ streaming, ứng dụng chat và các hệ thống mạng khác. Nó cung cấp các công cụ hiệu suất cao để xử lý nhiều tác vụ cùng lúc, giúp giảm thiểu tác động đến tài nguyên của hệ thống.

4.3 GIAO THỨC DIFFIE – HELLMAN

"Giao thức Diffie-Hellman là một phương pháp mật mã học được sử dụng để tạo ra một khoá bí mật chung giữa hai hoặc nhiều bên thông qua một kênh không an toàn. Phương pháp này đã định hình cơ sở cho việc thiết lập kết nối an toàn trong các hệ thống mạng và bảo mật thông tin trực tuyến.

Ý tưởng chính của giao thức này là cho phép các bên giao tiếp tạo ra một khoá bí mật chung mà không cần truyền bất kỳ thông tin về khoá bí mật qua kênh không an toàn" [6], [8]. Cách thức hoạt động của Diffie-Hellman như sau:

- Lựa chọn các tham số chia sẻ công khai: Hai bên tham gia giao tiếp (ví dụ là Alice và Bob) đồng ý về các tham số chung mà bất kỳ ai cũng có thể biết được. Tham số

này gồm số nguyên tố lớn p và một số nguyên g (thường là một số nguyên tố gốc).

- Tạo khoá bí mật cục bộ: Mỗi bên tạo ra một số nguyên bí mật của mình (đôi khi được gọi là số private key): a cho Alice và b cho Bob. Những số này chỉ được bên tạo ra biết.
- Tính toán khoá công khai: Mỗi bên tính toán khoá công khai của mình bằng cách sử dụng tham số chia sẻ công khai và số bí mật của mình. Alice tính $A = g^a \, mod \, b$, trong khi Bob tính $B = g^b \, mod \, p$.
- Trao đổi và tính toán khoá chung: Alice và Bob trao đổi khoá công khai của mình (tức là A và B). Sau đó, mỗi bên sử dụng khoá công khai của đối phương và số bí mật của mình để tính toán khoá bí mật chung. Alice tính $s = B^a \, mod \, p$, trong khi Bob tính $s = A^b \, mod \, p$. Khoá bí mật s này sẽ giống nhau ở cả hai bên.

4.4 TIÊU CHUẨN MÃ HÓA TIÊN TIẾN (AES)

"AES (Advanced Encryption Standard) là một thuật toán mã hóa khối (block cipher) phổ biến, thường được sử dụng trong các ứng dụng mã hóa và bảo mật dữ liệu. Được chính phủ Hoa Kỳ chọn làm tiêu chuẩn mã hóa chính thức, AES hiện đã trở thành một trong những thuật toán mã hóa được sử dụng rộng rãi nhất trên toàn cầu" [7], [8]. Cách hoạt động của AES như sau:

- **Khởi tạo:** Bắt đầu với một khóa được chọn và chuẩn bị dữ liệu cần mã hóa thành các khối cố định kích thước 128-bit.
- **SubBytes:** Mỗi byte trong khối dữ liệu được thay thế bằng một byte khác thông qua một bảng thay thế (S-box).
- **ShiftRows:** Các hàng trong khối dữ liệu được dịch chuyển sang trái theo một số lượng byte xác định.
- **MixColumns:** Các cột trong khối dữ liệu được kết hợp thông qua các phép nhân ma trận.
- **AddRoundKey:** Mỗi khối dữ liệu được kết hợp với một khóa con được tạo từ khóa chính.

4.5 HÀM BĂM

4.5.1 MD5

"MD5 (Message Digest Algorithm 5) là một trong những thuật toán băm (hash) được sử dụng rộng rãi để tạo ra một giá trị băm duy nhất (đôi khi được gọi là checksum hoặc hash value) từ một đầu vào dữ liệu. Nó được thiết kế để tạo ra một giá trị băm 128-bit, thường được biểu diễn bằng một chuỗi 32 ký tự hexa" [5], [8].

Tuy nhiên, cần lưu ý rằng MD5 đã bị các vấn đề về bảo mật phát hiện từ năm 1996 và các tấn công ngày càng phát triển dẫn đến việc MD5 không còn an toàn cho các ứng dụng bảo mật cao. Cách thức hoạt động của MD5:

- Băm dữ liệu: Đầu vào (dữ liệu cần băm) được chia thành các khối 512-bit và được xử lý theo từng khối.
- Thêm bit padding: Để đảm bảo rằng đầu vào có độ dài phù hợp, bit padding được thêm vào cuối dữ liệu.
- Giai đoạn nén: Mỗi khối dữ liệu được xử lý qua một loạt các vòng nén sử dụng hàm luân phiên (permutation function), hàm non-linear (non-linear function) và các phép lấy modulo.
- Tạo giá trị băm: Cuối cùng, sau khi tất cả các khối đã được xử lý, giá trị băm 128-bit được tạo ra từ các giá trị trung gian được tính toán trong quá trình nén.

4.5.2 SHA-256

Ban đầu, MD5 được sử dụng rộng rãi trong việc bảo mật mật khẩu và xác thực, cũng như trong việc kiểm tra tính toàn vẹn của các file. Tuy nhiên, do các vấn đề bảo mật đã được phát hiện, nên các thuật toán khác như SHA-256, SHA-3 đã được khuyến nghị thay thế MD5 trong các ứng dụng yêu cầu mức độ bảo mật cao hơn.

SHA-256 là một trong các thuật toán băm thuộc họ SHA-2 (Secure Hash Algorithm 2), được thiết kế để tạo ra một giá trị băm có độ dài 256-bit từ một đầu vào dữ liệu bất kỳ, đảm bảo tính duy nhất và khó khăn để trở lại dữ liệu ban đầu.

SHA-256, được phát triển bởi Cơ quan An ninh Quốc gia Hoa Kỳ (NSA), đã trở thành một trong những thuật toán băm được sử dụng rộng rãi nhất trong các ứng dụng bảo mật thông tin. Cách thức hoạt động của SHA-256 như sau:

- **Chuẩn bị đầu vào:** Đầu vào dữ liệu được chia thành các khối 512-bit.
- **Thêm bit padding:** Một số bit được thêm vào cuối dữ liệu để đảm bảo rằng kích thước cuối cùng của dữ liệu là 512-bit, theo định dạng của thuật toán.
- **Giai đoạn nén:** Các khối dữ liệu được xử lý qua một loạt các vòng nén sử dụng hàm luân phiên (permutation function), hàm non-linear (non-linear function), và các phép lấy modulo, tương tự như cách hoạt động của các thuật toán băm khác.
- **Tạo giá trị băm:** Cuối cùng, sau khi tất cả các khối đã được xử lý, giá trị băm 256-bit được tạo ra từ các giá trị trung gian được tính toán trong quá trình nén.

4.5.3 SHA-3

SHA-3 (Secure Hash Algorithm 3) là một thuật toán băm được thiết kế để tạo ra các giá trị băm từ các đầu vào dữ liệu bất kỳ. Nó là một trong những tiêu chuẩn bảo mật được công nhận quốc tế, được phát triển bởi hội đồng tiêu chuẩn hóa Quốc tế (NIST - National Institute of Standards and Technology) và được công bố vào năm 2015.

Thuật toán SHA-3 không chỉ đơn thuần là một phiên bản nâng cấp của SHA-2, mà là một thuật toán băm hoàn toàn mới được xây dựng từ các cơ sở thiết kế hoàn toàn khác biệt:

- **Kiến trúc hoàn toàn khác biệt:** SHA-3 không chia sẻ cùng kiến trúc hoặc cơ sở với SHA-2.
- **Khả năng chống tấn công:** SHA-3 được thiết kế để chống lại các phương pháp tấn công phổ biến hiện đại như collision attacks (tấn công va chạm).
- **Độ dài băm linh hoạt:** SHA-3 có thể tạo ra các giá trị băm với độ dài khác nhau, không giống như SHA-2 chỉ tạo ra các giá trị băm với độ dài cố định.
- **Tính hiệu quả và hiệu suất:** SHA-3 có khả năng chạy trên nhiều nền tảng khác nhau với tốc độ cao và hiệu suất tốt.

4.5.4 BLAKE2

Blake2 là một thuật toán băm hiện đại và linh hoạt, được phát triển nhằm tạo ra giá trị băm từ bất kỳ đầu vào dữ liệu nào. Đây là phiên bản nâng cấp của thuật toán Blake, do Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn và Christian Winnerlein phát triển vào năm 2013. Blake2 có những đặc điểm nổi bật sau:

- **Hiệu suất cao:** Blake2 được thiết kế để có hiệu suất cao trên nhiều nền tảng khác nhau, từ thiết bị nhúng đến máy tính cá nhân và các hệ thống có khả năng tính toán cao.
- **Đa năng và linh hoạt:** Blake2 hỗ trợ việc tạo ra các giá trị băm với độ dài khác nhau, từ 1 đến 64 byte.
- **Bảo mật cao:** Blake2 được thiết kế để đáp ứng các tiêu chí an ninh mật mã hiện đại, đặc biệt là tránh các vấn đề như collision attacks.
- **Hiệu quả và tiết kiệm tài nguyên:** Blake2 được thiết kế để tiết kiệm tài nguyên tính toán và bộ nhớ so với một số thuật toán băm khác.
- **Hỗ trợ nhiều dạng đầu vào:** Blake2 có khả năng xử lý cả dữ liệu có kích thước từ nhỏ đến lớn, từ tin nhắn văn bản đến file lớn.

Thuật toán Blake2 có hai biến thể chính:

- **Blake2b:** Được thiết kế cho các ứng dụng đòi hỏi độ dài băm lớn, với giá trị băm có độ dài từ 1 đến 64 byte.
- **Blake2s:** Thiết kế cho các ứng dụng yêu cầu giá trị băm có độ dài ngắn hơn, với giá trị băm từ 1 đến 32 byte.

4.6 BÀI TẬP THỰC HÀNH

4.6.1 Bài thực hành 01: Tạo Socket với AES và RSA

Tạo ứng dụng client-server giao tiếp thông qua cơ chế Socket và được mã hoá bằng mật mã AES và RSA.

Hướng dẫn:

- Clone Git repo về máy tính thực hành (Mở trong CMD):

```
git clone <đường dẫn git repo của sinh viên>
cd .\bmtt-nc-hutech-mssv\
code .
```

- Tại giao diện VS Code, mở Terminal, pull mới code về:

```
git pull origin main
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249> git pull origin main
From https://github.com/mrphuoc020597/bmtt-nc-hutech-1511060249
 * branch            main       -> FETCH_HEAD
Already up to date.
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249>
```

- Tách nhánh lab04 từ nhánh main.

```
git checkout -b lab04
```

- Tạo folder "lab-04". Trong folder "lab-04" tạo folder "aes_rsa_socket".
- Trong folder "aes_rsa_socket" tạo file "requirements.txt".

```
requirements.txt ×
lab-04 > aes_rsa_socket > requirements.txt
  1   pycryptodome
```

- Vào Terminal, chạy các lệnh sau:

```
cd .\lab-04\aes_rsa_socket\
pip install -r .\requirements.txt
```

- Trong folder "aes_rsa_socket" tạo file "server.py".

server.py

lab-04 > aes_rsa_socket > server.py > handle_client

```python
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import socket
import threading
import hashlib

# Initialize server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 12345))
server_socket.listen(5)

# Generate RSA key pair
server_key = RSA.generate(2048)

# List of connected clients
clients = []

# Function to encrypt message
def encrypt_message(key, message):
    cipher = AES.new(key, AES.MODE_CBC)
    ciphertext = cipher.encrypt(pad(message.encode(), AES.block_size))
    return cipher.iv + ciphertext

# Function to decrypt message
def decrypt_message(key, encrypted_message):
    iv = encrypted_message[:AES.block_size]
    ciphertext = encrypted_message[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted_message.decode()

```

```python
# Function to handle client connection
def handle_client(client_socket, client_address):
    print(f"Connected with {client_address}")

    # Send server's public key to client
    client_socket.send(server_key.publickey().export_key(format='PEM'))

    # Receive client's public key
    client_received_key = RSA.import_key(client_socket.recv(2048))

```

```
    # Generate AES key for message encryption
    aes_key = get_random_bytes(16)

    # Encrypt the AES key using the client's public key
    cipher_rsa = PKCS1_OAEP.new(client_received_key)
    encrypted_aes_key = cipher_rsa.encrypt(aes_key)
    client_socket.send(encrypted_aes_key)

    # Add client to the list
    clients.append((client_socket, aes_key))

    while True:
        encrypted_message = client_socket.recv(1024)
        decrypted_message = decrypt_message(aes_key, encrypted_message)
        print(f"Received from {client_address}: {decrypted_message}")

        # Send received message to all other clients
        for client, key in clients:
            if client != client_socket:
                encrypted = encrypt_message(key, decrypted_message)
                client.send(encrypted)

        if decrypted_message == "exit":
            break

    clients.remove((client_socket, aes_key))
    client_socket.close()
    print(f"Connection with {client_address} closed")

# Accept and handle client connections
while True:
    client_socket, client_address = server_socket.accept()
    client_thread = threading.Thread(target=handle_client, args=(client_socket,
    client_address))
    client_thread.start()
```

- Trong folder “aes_rsa_socket” tạo file “client.py”.

client.py

lab-04 > aes_rsa_socket > client.py > encrypt_message

```
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import socket
import threading
import hashlib

```

```python
# Initialize client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 12345))

# Generate RSA key pair
client_key = RSA.generate(2048)

# Receive server's public key
server_public_key = RSA.import_key(client_socket.recv(2048))

# Send client's public key to the server
client_socket.send(client_key.publickey().export_key(format='PEM'))

# Receive encrypted AES key from the server
encrypted_aes_key = client_socket.recv(2048)

# Decrypt the AES key using client's private key
cipher_rsa = PKCS1_OAEP.new(client_key)
aes_key = cipher_rsa.decrypt(encrypted_aes_key)

```

```python
# Function to encrypt message
def encrypt_message(key, message):
    cipher = AES.new(key, AES.MODE_CBC)
    ciphertext = cipher.encrypt(pad(message.encode(), AES.block_size))
    return cipher.iv + ciphertext

# Function to decrypt message
def decrypt_message(key, encrypted_message):
    iv = encrypted_message[:AES.block_size]
    ciphertext = encrypted_message[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted_message.decode()

# Function to receive messages from server
def receive_messages():
    while True:
        encrypted_message = client_socket.recv(1024)
        decrypted_message = decrypt_message(aes_key, encrypted_message)
        print("Received:", decrypted_message)

```

```python
# Start the receiving thread
receive_thread = threading.Thread(target=receive_messages)
receive_thread.start()

# Send messages from the client
while True:
    message = input("Enter message ('exit' to quit): ")
    encrypted_message = encrypt_message(aes_key, message)
    client_socket.send(encrypted_message)
    if message == "exit":
        break

# Close the connection when done
client_socket.close()
```

- Kiểm tra ứng dụng:
  - Bước 1: Split Terminal thành 3 cửa sổ. Tại cửa sổ 1 chạy file "server.py".

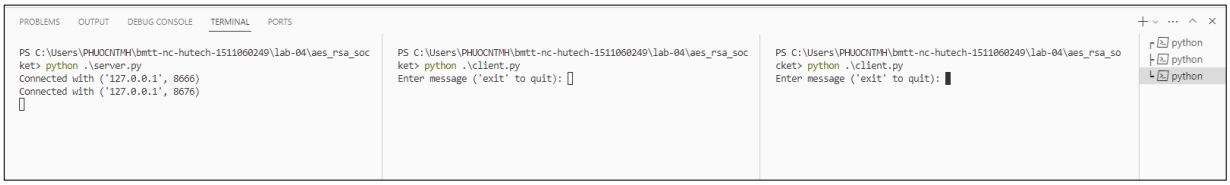
```
python .\server.py
```

  - Bước 2: Tại cửa sổ Terminal thứ 2, chạy file "client.py". Ở server hiển thị kết nối.

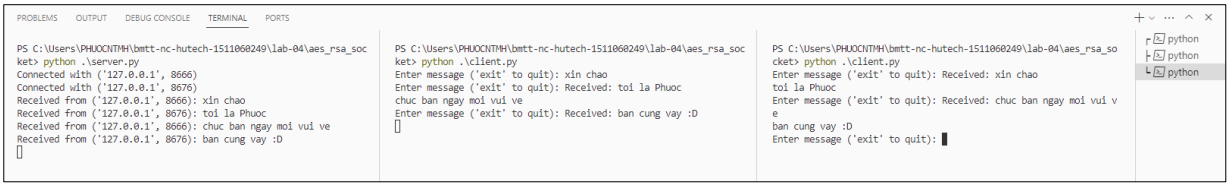
```
python .\client.py
```

  - Bước 3: Tại cửa sổ Terminal thứ 3, chạy file "client.py". Ở server hiển thị kết nối.

```
python .\client.py
```



- Bước 4: Nhập nội dung chat tại hai client và kiểm tra.



- Commit code:

```
git add .
git commit -m "[add] lab-04 aes_rsa_socket"
```

4.6.2 Bài thực hành 02: DH Key Pair

Tạo ứng dụng client-server sử dụng Giao thức Diffie-Hellman để tạo và chia sẻ khoá.

Hướng dẫn:

- Trong folder "lab-04" tạo folder "dh_key_pair". Trong folder "dh_key_pair" tạo file "requirements.txt".

requirements.txt

lab-04 > dh_key_pair > requirements.txt

```
1  cryptography
```

- Vào Terminal, chạy các lệnh sau:

```
cd .\lab-04\dh_key_pair\
pip install -r .\requirements.txt
```

- Trong folder "dh_key_pair" tạo file "server.py".

server.py

lab-04 > dh_key_pair > server.py > ...

```
1  from cryptography.hazmat.primitives.asymmetric import dh
2  from cryptography.hazmat.primitives import serialization
3
4  def generate_dh_parameters():
5      parameters = dh.generate_parameters(generator=2, key_size=2048)
6      return parameters
7
```

```python
def generate_server_key_pair(parameters):
    private_key = parameters.generate_private_key()
    public_key = private_key.public_key()
    return private_key, public_key

def main():
    parameters = generate_dh_parameters()
    private_key, public_key = generate_server_key_pair(parameters)

    with open("server_public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

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

- Trong folder “dh_key_pair” tạo file “client.py”.

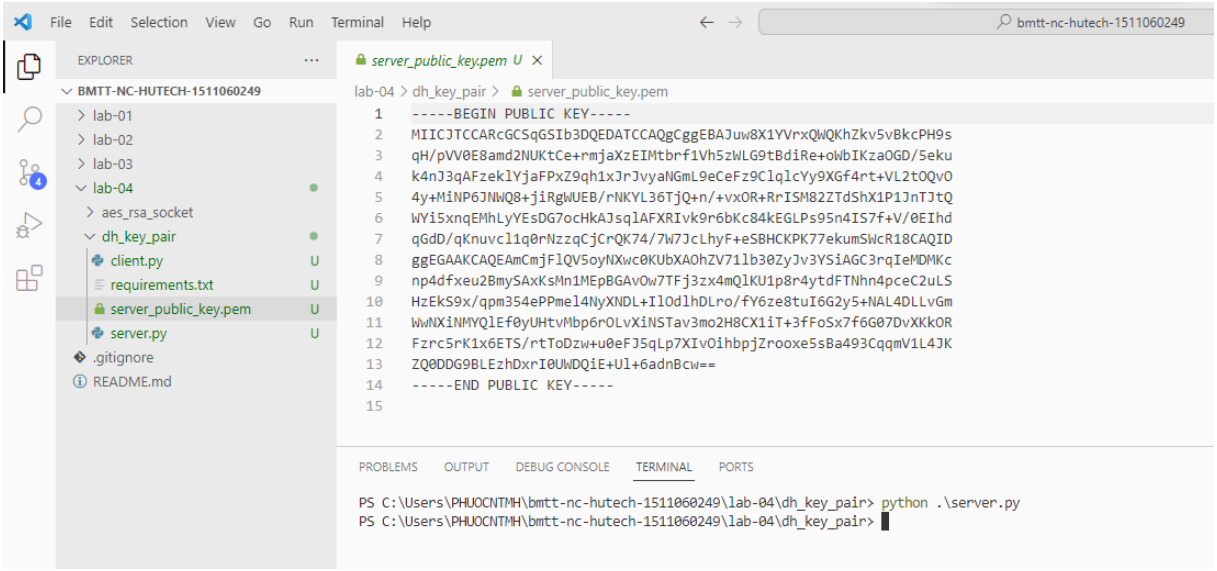
client.py

lab-04 > dh_key_pair > client.py > ...

```python
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes

def generate_client_key_pair(parameters):
    private_key = parameters.generate_private_key()
    public_key = private_key.public_key()
    return private_key, public_key

def derive_shared_secret(private_key, server_public_key):
    shared_key = private_key.exchange(server_public_key)
    return shared_key

def main():
    # Load server's public key
    with open("server_public_key.pem", "rb") as f:
        server_public_key = serialization.load_pem_public_key(f.read())

    parameters = server_public_key.parameters()
    private_key, public_key = generate_client_key_pair(parameters)

    shared_secret = derive_shared_secret(private_key, server_public_key)

    print("Shared Secret:", shared_secret.hex())

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

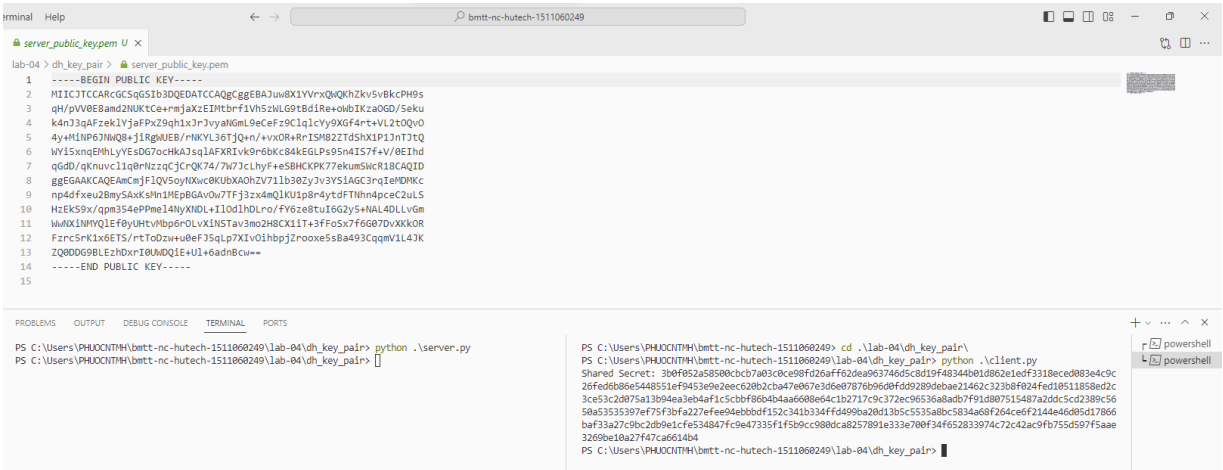
- Kiểm tra ứng dụng:
  - Bước 1: Chạy file "server.py". Chờ server tạo server_public_key và được lưu thành file server_public_key.pem.

```
python .\server.py
```



  - Bước 2: Chạy file "client.py". Client sẽ đọc và hiển thị nội dung thông điệp bí mật được chia sẻ.

```
python .\client.py
```



- Commit code:

```
git add .
git commit -m "[add] lab-04 dh_key_pair"
```

4.6.3 Bài thực hành 03: Hàm băm

Tạo ứng dụng dòng lệnh để thực hiện các hàm băm: MD5, SHA-256, SHA-3, Blake2.

Hướng dẫn:

- Trong folder "lab-04" tạo folder "hash". Trong folder "hash" tạo file "md5_hash.py" để thực hiện hàm băm MD5 không dùng thư viện.

md5_hash.py

lab-04 > hash > md5_hash.py > md5

```python
def left_rotate(value, shift):
    return ((value << shift) | (value >> (32 - shift))) & 0xFFFFFFFF
```

```python
def md5(message):
    # Khởi tạo các biến ban đầu
    a = 0x67452301
    b = 0xEFCDAB89
    c = 0x98BADCFE
    d = 0x10325476

    # Tiền xử lý chuỗi văn bản
    original_length = len(message)
    message += b'\x80'
    while len(message) % 64 != 56:
        message += b'\x00'
    message += original_length.to_bytes(8, 'little')

    # Chia chuỗi thành các block 512-bit
    for i in range(0, len(message), 64):
        block = message[i:i+64]

        words = [int.from_bytes(block[j:j+4], 'little') for j in range(0, 64, 4)]

        a0, b0, c0, d0 = a, b, c, d

        # Vòng lặp chính của thuật toán MD5
        for j in range(64):
            if j < 16:
                f = (b & c) | ((~b) & d)
                g = j
            elif j < 32:
                f = (d & b) | ((~d) & c)
                g = (5*j + 1) % 16
```

```
            elif j < 48:
                f = b ^ c ^ d
                g = (3*j + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7*j) % 16

            temp = d
            d = c
            c = b
            b = b + left_rotate((a + f + 0x5A827999 + words[g]) & 0xFFFFFFFF, 3)
            a = temp

        a = (a + a0) & 0xFFFFFFFF
        b = (b + b0) & 0xFFFFFFFF
        c = (c + c0) & 0xFFFFFFFF
        d = (d + d0) & 0xFFFFFFFF

    return '{:08x}{:08x}{:08x}{:08x}'.format(a, b, c, d)

input_string = input("Nhập chuỗi cần băm: ")
md5_hash = md5(input_string.encode('utf-8'))

print("Mã băm MD5 của chuỗi '{}' là: {}".format(input_string, md5_hash))
```

- Kiểm tra bằng cách chạy file "md5_hash.py".

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\md5_hash.py
Nhập chuỗi cần băm: hutech university
Mã băm MD5 của chuỗi 'hutech university' là: d78961379bf78cd5b3f89000ca12f01a
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\md5_hash.py
Nhập chuỗi cần băm: nguyen trong minh hong phuoc
Mã băm MD5 của chuỗi 'nguyen trong minh hong phuoc' là: a61ef8b56d9fe0dd786df7aded16ce13
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash>
```

- Trong folder "hash" tạo file "md5_library.py" để thực hiện hàm băm MD5 sử dụng thư viện có sẵn.

md5_library.py

lab-04 > hash > md5_library.py > ...

```
import hashlib

def calculate_md5(input_string):
    md5_hash = hashlib.md5()
    md5_hash.update(input_string.encode('utf-8'))
    return md5_hash.hexdigest()

input_string = input("Nhập chuỗi cần băm: ")
md5_hash = calculate_md5(input_string)

print("Mã băm MD5 của chuỗi '{}' là: {}".format(input_string, md5_hash))
```

- Kiểm tra bằng cách chạy file "md5_library.py".

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\md5_library.py
Nhập chuỗi cần băm: cong nghe thong tin hutech
Mã băm MD5 của chuỗi 'cong nghe thong tin hutech' là: fc727295ede53a235489a0b62a6be8cd
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\md5_library.py
Nhập chuỗi cần băm: xin chao cac ban toi la sinh vien hutech
Mã băm MD5 của chuỗi 'xin chao cac ban toi la sinh vien hutech' là: c8dd44c9e891eb4439c1e4e19873cf86
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash>
```

- Trong folder "hash" tạo file "sha-256.py" để thực hiện hàm băm SHA-256 sử dụng thư viện có sẵn.

sha-256.py

lab-04 > hash > sha-256.py > ...

```python
import hashlib

def calculate_sha256_hash(data):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(data.encode('utf-8'))  # Chuyển đổi dữ liệu thành bytes và cập nhật vào đối tượng hash
    return sha256_hash.hexdigest()  # Trả về biểu diễn hex chuỗi hash

data_to_hash = input("Nhập dữ liệu để hash bằng SHA-256: ")
hash_value = calculate_sha256_hash(data_to_hash)
print("Giá trị hash SHA-256:", hash_value)
```

- Kiểm tra bằng cách chạy file "sha-256.py".

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\sha-256.py
Nhập dữ liệu để hash bằng SHA-256: hutech university
Giá trị hash SHA-256: f36e7b18473931d91715be3b1dd3a5bdd2fc1bc53a8ed867440d4b29c1905506
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\sha-256.py
Nhập dữ liệu để hash bằng SHA-256: nguyen trong minh hong phuoc
Giá trị hash SHA-256: 1b446456b7f78174c76e763437854d5d0f3ac7054c0023546d00f1a867758564
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash>
```

- Trong folder "hash" tạo file "sha-3.py" để thực hiện hàm băm SHA-3 sử dụng thư viện có sẵn.

sha-3.py

lab-04 > hash > sha-3.py > ...

```python
from Crypto.Hash import SHA3_256

def sha3(message):
    sha3_hash = SHA3_256.new()
    sha3_hash.update(message)
    return sha3_hash.digest()

def main():
    text = input("Nhập chuỗi văn bản: ").encode('utf-8')
    hashed_text = sha3(text)

    print("Chuỗi văn bản đã nhập:", text.decode('utf-8'))
    print("SHA-3 Hash:", hashed_text.hex())

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

- Kiểm tra bằng cách chạy file "sha-3.py".

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\sha-3.py
Nhập chuỗi văn bản: hutech university
Chuỗi văn bản đã nhập: hutech university
SHA-3 Hash: f94e5d66bef9f7be64c5a78c781f6b8c055715c0d8628999790869a283866077
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\sha-3.py
Nhập chuỗi văn bản: nguyen trong minh hong phuoc
Chuỗi văn bản đã nhập: nguyen trong minh hong phuoc
SHA-3 Hash: f6da21fad86c5bae1bdc05c0197dd1f7e756d819896af338fd38e4243a1dd9af
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash>
```

- Trong foler "hash" tạo file "blake2.py" để thực hiện hàm băm Blake2 sử dụng thư viện có sẵn.

blake2.py

lab-04 > hash > blake2.py > ...

```python
import hashlib

def blake2(message):
    blake2_hash = hashlib.blake2b(digest_size=64)
    blake2_hash.update(message)
    return blake2_hash.digest()

def main():
    text = input("Nhập chuỗi văn bản: ").encode('utf-8')
    hashed_text = blake2(text)

    print("Chuỗi văn bản đã nhập:", text.decode('utf-8'))
    print("BLAKE2 Hash:", hashed_text.hex())

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

- Kiểm tra bằng cách chạy file "blake2.py".

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\blake2.py
Nhập chuỗi văn bản: hutech university
Chuỗi văn bản đã nhập: hutech university
BLAKE2 Hash: 12d7347881b1e8da4bd4bca2832b25f3551e0f2571201febbd1572c41169e0d08a0df343a2950acdf1884089a69f8b4c3eedf526f3a926cd8d465a2594925e2d
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash> python .\blake2.py
Nhập chuỗi văn bản: nguyen trong minh hong phuoc
Chuỗi văn bản đã nhập: nguyen trong minh hong phuoc
BLAKE2 Hash: 9545f211b44f21e17630e9a4ef642bc626d3997541fcde4c98ec4f8bc69b1e69b244d5274b31de6b166dd8604faee75299f76b3632392817b25ae46cc2ee9d40
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-04\hash>
```

- Commit code:

```
git add .
git commit -m "[add] lab-04 hash"
```

4.6.4 Bài thực hành 04: WebSocket và Tornado

Sử dụng Tornado để tạo một WebSocket cho chương trình giao tiếp client-server (streaming data), cứ mỗi 3 giây server sẽ gửi thông tin là tên một loại trái cây, các client sẽ nhận thông tin này và in ra màn hình.

Hướng dẫn:

- Trong folder "lab-04" tạo folder "websocket". Trong folder "websocket" tạo file "requirements.txt".

```
requirements.txt ×
lab-04 > websocket > requirements.txt
1  tornado
```

- Vào Terminal, chạy các lệnh sau:

```
cd .\lab-04\websocket\
pip install -r .\requirements.txt
```

- Trong folder "websocket" tạo file "server.py".

server.py

lab-04 > websocket > server.py > main

```python
import random
import tornado.ioloop
import tornado.web
import tornado.websocket

class WebSocketServer(tornado.websocket.WebSocketHandler):
    clients = set()

    def open(self):
        WebSocketServer.clients.add(self)

    def on_close(self):
        WebSocketServer.clients.remove(self)

    @classmethod
    def send_message(cls, message: str):
        print(f"Sending message {message} to {len(cls.clients)} client(s).")
        for client in cls.clients:
            client.write_message(message)

class RandomWordSelector:
    def __init__(self, word_list):
        self.word_list = word_list

    def sample(self):
        return random.choice(self.word_list)

def main():
    app = tornado.web.Application(
        [(r"/websocket/", WebSocketServer)],
        websocket_ping_interval=10,
        websocket_ping_timeout=30,
    )
    app.listen(8888)

    io_loop = tornado.ioloop.IOLoop.current()

    word_selector = RandomWordSelector(['apple', 'banana', 'orange', 'grape',
    'melon'])

    periodic_callback = tornado.ioloop.PeriodicCallback(
        lambda: WebSocketServer.send_message(word_selector.sample()), 3000
    )
    periodic_callback.start()

    io_loop.start()

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

- Trong folder “websocket” tạo file “client.py”.

client.py

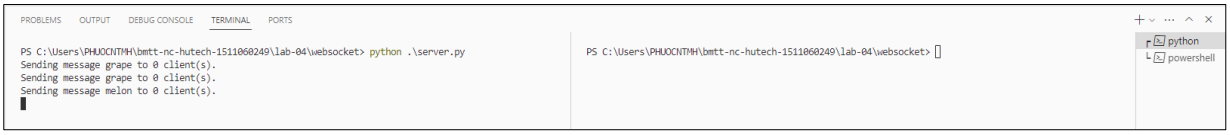
lab-04 > websocket > client.py > WebSocketClient > on_message

```python
import tornado.ioloop
import tornado.websocket

class WebSocketClient:
    def __init__(self, io_loop):
        self.connection = None
        self.io_loop = io_loop

    def start(self):
        self.connect_and_read()

    def stop(self):
        self.io_loop.stop()

    def connect_and_read(self):
        print("Reading...")
        tornado.websocket.websocket_connect(
            url=f"ws://localhost:8888/websocket/",
            callback=self.maybe_retry_connection,
            on_message_callback=self.on_message,
            ping_interval=10,
            ping_timeout=30,
        )

    def maybe_retry_connection(self, future) -> None:
        try:
            self.connection = future.result()
        except:
            print("Could not reconnect, retrying in 3 seconds...")
            self.io_loop.call_later(3, self.connect_and_read)

    def on_message(self, message):
        if message is None:
            print("Disconnected, reconnecting...")
            self.connect_and_read()
            return

        print(f"Received word from server: {message}")

        self.connection.read_message(callback=self.on_message)

def main():
    io_loop = tornado.ioloop.IOLoop.current()

    client = WebSocketClient(io_loop)
    io_loop.add_callback(client.start)

    io_loop.start()

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

- Kiểm tra ứng dụng:

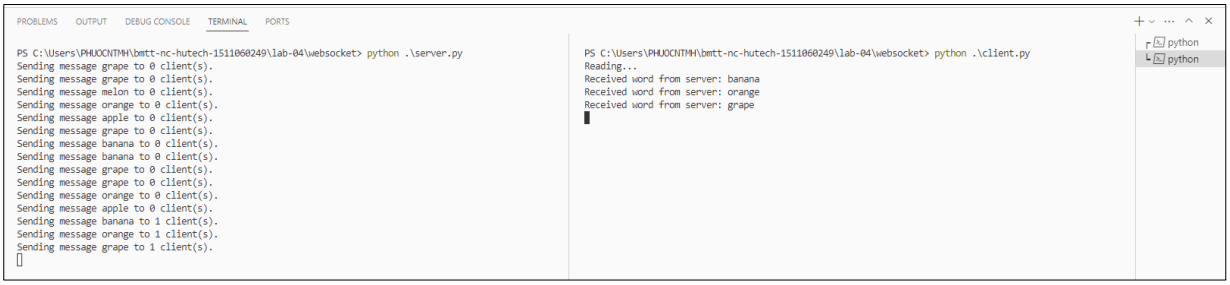
  - Bước 1: Split Terminal, chạy file "server.py".

```
python .\server.py
```

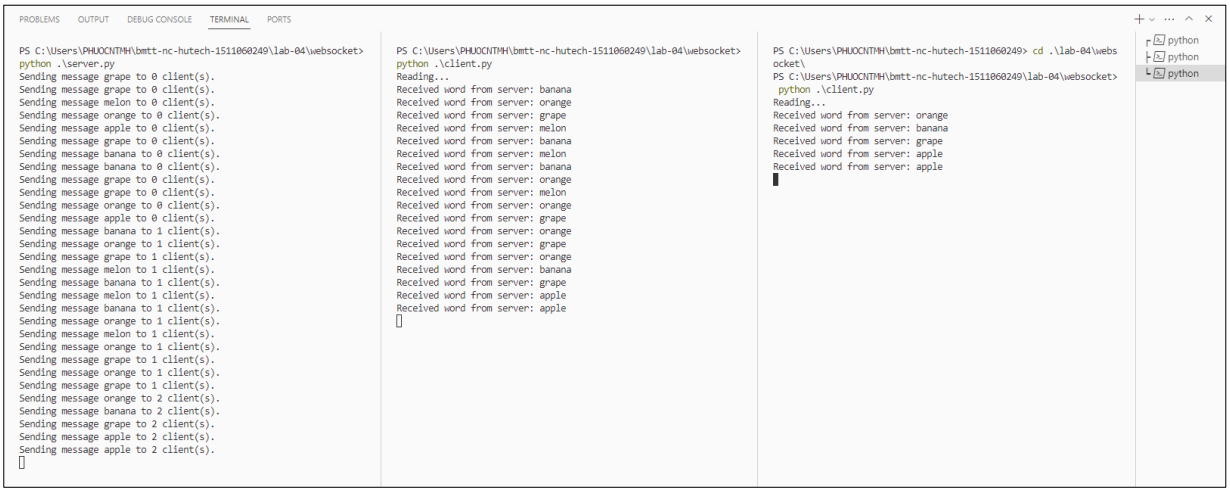


  - Bước 2: Chạy file "client.py".

```
python .\client.py
```



  Bước 3: Chạy thêm một "client.py".



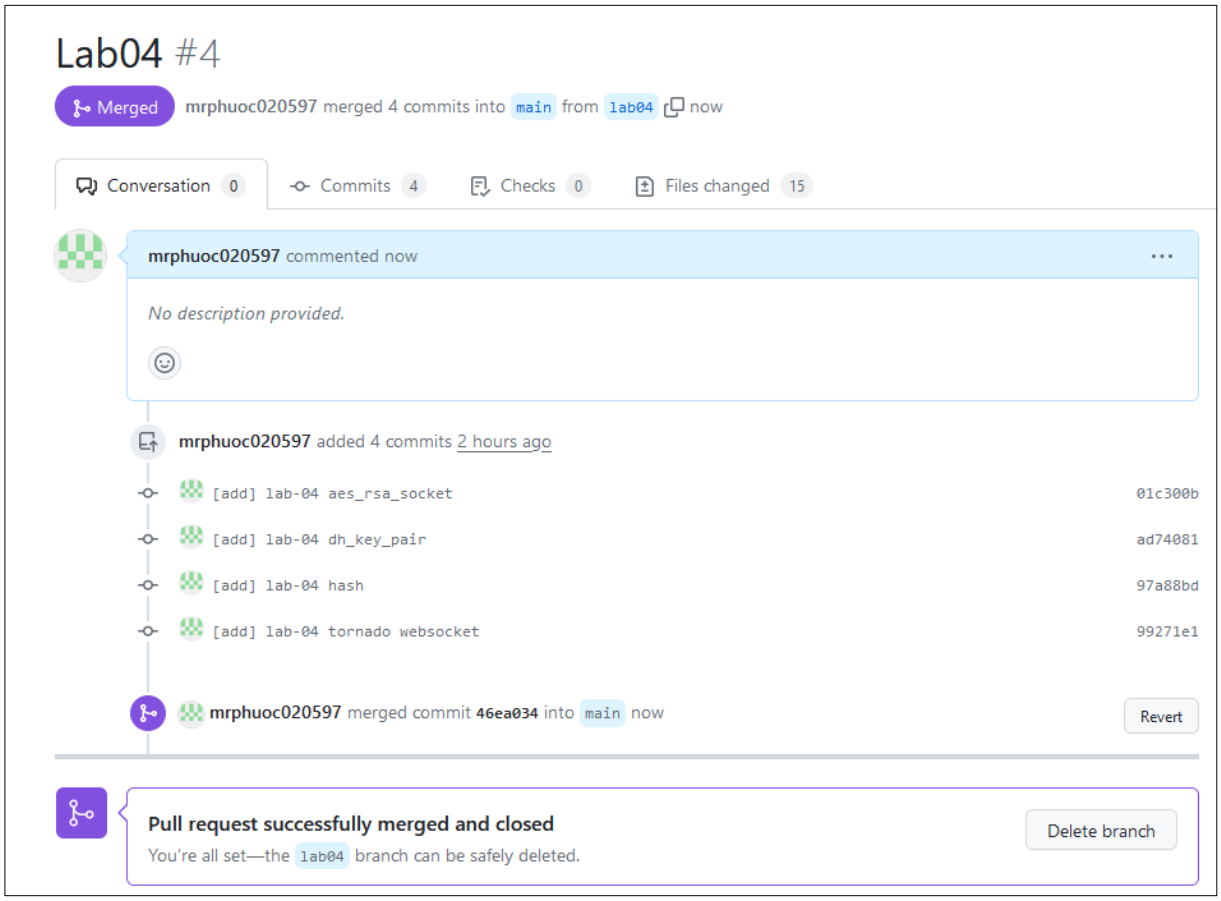
- Commit code:

```
git add .
git commit -m "[add] lab-04 tornado websocket"
```

- Push các thay đổi lên remote repo:

```
git push origin lab04
```

- Tiến hành tạo PR từ nhánh lab04 về nhánh main. Review, kiểm tra đủ số lượng file, sau đó tiến hành "Merge pull request".



4.7 BÀI TẬP MỞ RỘNG

- **Câu 01:** Xây dựng một giao diện UI desktop cho Bài thực hành 01.
- **Câu 02:** Xây dựng một giao diện UI desktop cho Bài thực hành 02.
- **Câu 03:** Xây dựng một giao diện UI desktop cho Bài thực hành 03.
- **Câu 04:** Sử dụng Tornado để tạo một WebSocket cho chương trình giao tiếp client-server. Client sẽ gửi thông điệp đến server. Server sẽ mã hoá thông điệp đó bằng AES và gửi kết quả về client.