Sender

```python
import os
import base64
import hashlib
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import requests
import random
from cryptography.hazmat.primitives import hashes

# Your chosen values for p and g
p = 23
g = 5

def generate_dh_private_key():
    # Generate a random Diffie-Hellman private key
    dh_private_key = random.randint(1, p-1)
    return dh_private_key

def generate_dh_public_key(private_key):
    # Generate Diffie-Hellman public key
    dh_public_key = pow(g, private_key, p)
    return dh_public_key

def pad(plaintext):
    # Add PKCS7 padding
    padding_len = 16 - len(plaintext) % 16
    return plaintext + bytes([padding_len]) * padding_len

def encrypt_file(file_path, session_key):
    # Open and read the file
    with open(file_path, 'rb') as file:
        plaintext = file.read()
```

```
# Add padding
    plaintext = pad(plaintext)

    # Generate a random IV
    iv = os.urandom(16)

    # Create AES cipher
    cipher = Cipher(algorithms.AES(session_key), modes.CBC(iv))
    encryptor = cipher.encryptor()

    # Encrypt the file
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()

    return iv, ciphertext

def encrypt_session_key(session_key, public_key):
    rsa_public_key = serialization.load_pem_public_key(
        public_key.encode(),
    )
    encrypted_session_key = rsa_public_key.encrypt(
        session_key,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return encrypted_session_key
```

نسوي pad

نستخدمه في CBC

النوع

نعطيها

كيف حنقسم البلوك وكم واحد

بسوي التكريبت ويحطه في cip

يرجع عشان يعرف كم انقسم

reseiver

يقرأ الملف فايل حقه الريسيفر

يحطه جوة المتغير

هنا حنشفر السيشن كي

reseiver

اللي نبغى نشفره

نوع البادنق

نوع الهاش

التشفير

يرجع

```python
def send_file(iv, encrypted_file, encrypted_session_key, dh_public_key):
    # Convert the bytes to base64
    b64_file = base64.b64encode(encrypted_file).decode('utf-8')
    b64_session_key = base64.b64encode(encrypted_session_key).decode('utf-8')
    b64_iv = base64.b64encode(iv).decode('utf-8')

    # Send the file
    requests.post('http://localhost:5000/receive', json={
        'file': b64_file,
        'session_key': b64_session_key,
        'public_key': dh_public_key,
        'p': p,
        'g': g,
        'iv': b64_iv,
    })
```

main

```python
# Generate the sender's Diffie-Hellman private and public keys
sender_dh_private_key = generate_dh_private_key()
sender_dh_public_key = generate_dh_public_key(sender_dh_private_key)

#print(f"Sender's Diffie-Hellman public key is: {sender_dh_public_key}")

# Generate the recipient's Diffie-Hellman private and public keys
recipient_dh_private_key = generate_dh_private_key()
recipient_dh_public_key = generate_dh_public_key(recipient_dh_private_key)

#print(f"Recipient's Diffie-Hellman public key is: {recipient_dh_public_key}")
```

```
# Generate the shared secret key
shared_secret_key = pow(recipient_dh_public_key, sender_dh_private_key, p)

# Hash the shared secret key to create the session key
session_key = hashlib.sha256(str(shared_secret_key).encode()).digest()

# Get the file path
file_path = input("Enter the path of the file to be transferred: ")

# Encrypt the file
iv, encrypted_file = encrypt_file(file_path, session_key)

# Get the recipient's public key
recipient_public_key_path = input("Enter the path to the recipient's public key:
")
with open(recipient_public_key_path, 'r') as key_file:
    recipient_public_key = key_file.read()

# Encrypt the session key
encrypted_session_key = encrypt_session_key(session_key, recipient_public_key)

# Send the file
send_file(iv, encrypted_file, encrypted_session_key, sender_dh_public_key)
```

Key

RSA Key

```python
name = str(input("For who will this belong to?"))

def generate_rsa_key():
    from cryptography.hazmat.primitives.asymmetric import rsa
    from cryptography.hazmat.primitives import serialization

    # Generate a new RSA private key
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )

    # Serialize the private key
    private_key_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    )

    # Generate the public key from the private key
    public_key = private_key.public_key()

    # Serialize the public key
    public_key_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )

    return private_key_pem, public_key_pem

# Generate a new RSA key pair
private_key_pem, public_key_pem = generate_rsa_key()

# Save the private key to a file
with open('D:/Downloads/keys/private_key_' +name+'.pem', 'wb') as f:
    f.write(private_key_pem)

with open('D:/Downloads/keys/public_key_' +name+'.pem', 'wb') as f:
    f.write(public_key_pem)
```

Recever

```python
from flask import Flask, request, jsonify
import base64
import hashlib
import random
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

app = Flask(__name__)

def generate_dh_private_key(p):
    dh_private_key = random.randint(1, p-1)
    return dh_private_key

def decrypt_file(encrypted_file, session_key, iv):
    cipher = Cipher(algorithms.AES(session_key), modes.CBC(iv))
    decryptor = cipher.decryptor()
    decrypted_file = decryptor.update(encrypted_file) + decryptor.finalize()
    return unpad(decrypted_file)

# Remove PKCS7 padding from the decrypted file
def unpad(decrypted_file):
    padding_length = decrypted_file[-1]
    return decrypted_file[:-padding_length]
```

```python
# RSA decryption of the session key
def decrypt_session_key(encrypted_session_key, private_key):
    decrypted_session_key = private_key.decrypt(
        encrypted_session_key,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_session_key

# Generate Diffie-Hellman public key
def generate_dh_public_key(p, g, private_key):
    public_key = pow(g, private_key, p)
    return public_key

@app.route('/receive', methods=['POST'])
def receive_file():
    data = request.get_json()

    # Extract the data
    encrypted_file = base64.b64decode(data['file'])
    encrypted_session_key = base64.b64decode(data['session_key'])
    sender_public_key = data['public_key']
    p = data['p']
    g = data['g']
    iv = base64.b64decode(data['iv'])

    # Load RSA private key from file
    private_key_path = input("Enter the path to your RSA private key: ")
    with open(private_key_path, 'rb') as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None
        )
```

```
    # Generate Diffie-Hellman private key
    dh_private_key = generate_dh_private_key(p)

    # Generate Diffie-Hellman public key
    dh_public_key = generate_dh_public_key(p, g, dh_private_key)

    # Generate the shared secret key
    shared_secret_key = pow(sender_public_key, dh_private_key, p)

    # Generate the session key
    session_key = hashlib.sha256(str(shared_secret_key).encode()).digest()

    # Decrypt the session key with RSA
    decrypted_session_key = decrypt_session_key(encrypted_session_key, private_key)


    # Decrypt the file with AES
    decrypted_file = decrypt_file(encrypted_file, decrypted_session_key, iv)

    # Save the decrypted file
with open('D:/Downloads/decrypted_file.txt', 'wb') as file:
    file.write(decrypted_file)

    print("File received and decrypted successfully!")
    print("Here are the contents:")
with open('D:/Downloads/decrypted_file.txt', 'r') as f:
    print(f.read())
    return jsonify({'status': 'success'}), 200


if __name__ == "__main__":
    app.run(host='localhost', port=5000)
```