```
import time
import os
import random
import string
from Crypto.Cipher import AES, DES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
```

These are libraries before. were used to perform encryption and decryption using three algorithm AES, DES and rsa.

```
# Function to generate random text content
def generate_random_text(size):
    return ''.join(random.choice(string.ascii_letters) for _ in
range(size))

# Function to create sample files
def create_sample_file(file_name, file_size):
    with open(file_name, 'wb') as file:
        file.write(os.urandom(file_size))

# Generate keys for AES, DES, and RSA
def generate_aes_key():
    return get_random_bytes(16)

def generate_des_key():
    return get_random_bytes(8)

def generate_rsa_key_pair():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key

# Function to encrypt and decrypt using AES
def encrypt_aes(file_path, key):
    cipher = AES.new(key, AES.MODE_EAX)
    with open(file_path, 'rb') as file:
        data = file.read()
    ciphertext, tag = cipher.encrypt_and_digest(data)
    return ciphertext, tag, cipher.nonce

def decrypt_aes(ciphertext, tag, nonce, key):
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
    return data

# Function to encrypt and decrypt using DES
def encrypt_des(file_path, key):
    iv = get_random_bytes(8)
    cipher = DES.new(key, DES.MODE_OFB, iv)
    with open(file_path, 'rb') as file:
        data = file.read()
    ciphertext = cipher.encrypt(data)
    return ciphertext, iv

def decrypt_des(ciphertext, iv, key):
    cipher = DES.new(key, DES.MODE_OFB, iv)
    data = cipher.decrypt(ciphertext)
    return data
```

This generates random data to put inside file.

This loop add data in the files created before

key generation for AES.

key generation for DES

Public and private key generation of RSA.

Functions for encryption and decryption of plaintext and ciphertext for AES.

Function of encryption and decryption of plain text and ciphertext for DES.

```
# Function to encrypt and decrypt using RSA (Hybrid Encryption)
def encrypt_rsa(file_path, public_key_path):
    with open(public_key_path, 'rb') as file:
        public_key = RSA.import_key(file.read())
    with open(file_path, 'rb') as file:
        data = file.read()
    cipher = PKCS1_OAEP.new(public_key)
    ciphertext = cipher.encrypt(data)
    return ciphertext

def decrypt_rsa(ciphertext, private_key_path):
    with open(private_key_path, 'rb') as file:
        private_key = RSA.import_key(file.read())
    cipher = PKCS1_OAEP.new(private_key)
    data = cipher.decrypt(ciphertext)
    return data
```

Function to encrypt & decrypt of RSA for plain text and cipher text.

```
# Create sample files for testing
file_sizes = {
    'small': 1024,          # 1KB
    'medium': 10 * 1024 * 1024,  # 10MB
    'large': 50 * 1024 * 1024    # 50MB
}
```

This code defines the sizes of small, medium and large given in assignment.

```
file_info = []

for size, size_bytes in file_sizes.items():
    for file_type in ['txt', 'exe', 'zip']:
        file_name = f'Shayan_Hassan_20K1873_{size}_{file_type}.
{file_type}'
        create_sample_file(file_name, size_bytes)
        file_info.append({'name': file_name, 'size': size, 'type':
file_type})
```

Creation of files and naming them with my name and roll number.

```
# Generate keys for RSA
private_key, public_key = generate_rsa_key_pair()
with open('private_key.pem', 'wb') as private_key_file:
    private_key_file.write(private_key)
with open('public_key.pem', 'wb') as public_key_file:
    public_key_file.write(public_key)
```

Saving the private and public keys in the files.

```
# Encryption and decryption timing for all files and algorithms
for info in file_info:
    file_path = info['name']

    # Encryption and decryption timing for AES
    aes_key = generate_aes_key()
    start_time = time.time()
    ciphertext_aes, tag_aes, nonce_aes = encrypt_aes(file_path,
```

Applying AES algorithm on all the files created.

```
aes_key)
    decrypted_data_aes = decrypt_aes(ciphertext_aes, tag_aes,
nonce_aes, aes_key)
    end_time = time.time()
    print(f"AES Encryption and Decryption Time for {info['size']}
{info['type']} file: {end_time - start_time} seconds")


    # Encryption and decryption timing for DES
    des_key = generate_des_key()
    start_time = time.time()
    ciphertext_des, iv_des = encrypt_des(file_path, des_key)
    decrypted_data_des = decrypt_des(ciphertext_des, iv_des, des_key)
    end_time = time.time()
    print(f"DES Encryption and Decryption Time for {info['size']}
{info['type']} file: {end_time - start_time} seconds")


    # Encryption and decryption timing for RSA (Hybrid Encryption)
    start_time = time.time()

    # Generate a random AES key for each file
    aes_key = generate_aes_key()

    # Encryption using AES
    cipher_aes = AES.new(aes_key, AES.MODE_EAX)
    with open(file_path, 'rb') as file:
        plaintext = file.read()
    ciphertext_rsa_aes, tag_rsa_aes, nonce_aes =
encrypt_aes(file_path, aes_key)

    # Decryption using AES
    decrypted_data_rsa_aes = decrypt_aes(ciphertext_rsa_aes,
tag_rsa_aes, nonce_aes, aes_key)
    end_time = time.time()
    print(f"RSA Encryption and Decryption Time for {info['size']}
{info['type']} file: {end_time - start_time} seconds")
```

Applying DES algorithm on files.

RSA applied on all the files and time calculation on them.

```
AES Encryption and Decryption Time for small txt file:
0.004728555679321289 seconds
DES Encryption and Decryption Time for small txt file:
0.001458883285522461 seconds
RSA Encryption and Decryption Time for small txt file:
0.0018734931945800781 seconds
AES Encryption and Decryption Time for small exe file:
0.001425027847290039 seconds
DES Encryption and Decryption Time for small exe file:
0.0007178783416748047 seconds
RSA Encryption and Decryption Time for small exe file:
```

outputs of all files.

```
0.001603841781616211 seconds
AES Encryption and Decryption Time for small zip file:
0.001277923583984375 seconds
DES Encryption and Decryption Time for small zip file:
0.0006647109985351562 seconds
RSA Encryption and Decryption Time for small zip file:
0.0017652511596679688 seconds
AES Encryption and Decryption Time for medium txt file:
0.1157078742980957 seconds
DES Encryption and Decryption Time for medium txt file:
0.3674483299255371 seconds
RSA Encryption and Decryption Time for medium txt file:
0.12756991386413574 seconds
AES Encryption and Decryption Time for medium exe file:
0.11881399154663086 seconds
DES Encryption and Decryption Time for medium exe file:
0.35589098930358887 seconds
RSA Encryption and Decryption Time for medium exe file:
0.13309764862060547 seconds
AES Encryption and Decryption Time for medium zip file:
0.15544342994689941 seconds
DES Encryption and Decryption Time for medium zip file:
0.5263614654541016 seconds
RSA Encryption and Decryption Time for medium zip file:
0.144758939743042 seconds
AES Encryption and Decryption Time for large txt file:
0.7149679660797119 seconds
DES Encryption and Decryption Time for large txt file:
1.8789973258972168 seconds
RSA Encryption and Decryption Time for large txt file:
0.9827449321746826 seconds
AES Encryption and Decryption Time for large exe file:
0.9575834274291992 seconds
DES Encryption and Decryption Time for large exe file:
3.9189627170562744 seconds
RSA Encryption and Decryption Time for large exe file:
0.7019026279449463 seconds
AES Encryption and Decryption Time for large zip file:
0.6485660076141357 seconds
DES Encryption and Decryption Time for large zip file:
1.8764135837554932 seconds
RSA Encryption and Decryption Time for large zip file:
0.6585428714752197 seconds
```

outputs of all files.

```
!pip install pycryptodome

Collecting pycryptodome
  Downloading pycryptodome-3.19.0-cp35-abi3-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 14.1 MB/s eta
```

installed library to use libraries imported in code.