**Annexe**

**Le voici la version 3 du code python implémentant l’algorithme RSA, en version corrigée, améliorée et optimisée :**

# Importing the random library to generate random numbers

import random

# Function to calculate the Greatest Common Divisor (GCD) of two numbers

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

# Function to calculate the modular inverse of a number modulo m

def modinv(a, m):

m0, x0, x1 = m, 0, 1

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

return x1 + m0 if x1 < 0 else x1

# Function to test if a number is prime (using the Miller-Rabin primality test)

def is\_prime(n, k=5):

if n <= 1:

return False

if n <= 3:

return True

if n % 2 == 0:

return False

d = n - 1

s = 0

while d % 2 == 0:

d //= 2

s += 1

for \_ in range(k):

a = random.randint(2, n - 1)

x = pow(a, d, n)

if x == 1 or x == n - 1:

continue

for \_ in range(s - 1):

x = pow(x, 2, n)

if x == n - 1:

break

else:

return False

return True

# Function to generate a potential prime number candidate

def generate\_prime\_candidate(length):

p = random.getrandbits(length)

p |= (1 << length - 1) | 1

return p

# Function to generate a prime number close to the specified length

def generate\_prime(length):

p = generate\_prime\_candidate(length)

while not is\_prime(p):

p = generate\_prime\_candidate(length)

return p

# Function to generate RSA keys with a specified key length

def generate\_keys(key\_length):

p = generate\_prime(key\_length // 4)

q = generate\_prime(key\_length // 4)

n = p \* q

phi = (p - 1) \* (q - 1)

e = random.randint(2, phi) # Start from 2 to ensure coprimality with phi

while gcd(e, phi) != 1:

e = random.randint(2, phi)

d = modinv(e, phi)

return ((e, n), (d, n))

# Function to encrypt a message

def encrypt(m, public\_key):

e, n = public\_key

return pow(m, e, n)

# Function to decrypt a message

def decrypt(c, private\_key):

d, n = private\_key

return pow(c, d, n)

# Function to encode a message into a list of integers representing the ASCII codes of characters

def encode\_message(message):

return [ord(char) for char in message]

# Function to decode a message encoded into a string of characters

def decode\_message(encoded\_message):

return ''.join(chr(char) for char in encoded\_message)

# This part of the Python program that implements effectively the RSA algorithm, displays the presentation of that Python program (that takes charge of the right choosing of the offered options described below, for the standard key sizes)

print("The present Python program will demonstrate the effectiveness of a right implementation of the RSA asymmetric encryption and decryption algorithm. For that purpose, the implementation of that asymmetric algorithm RSA uses a certain standard key sizes and standard message length determined by international organisations around the world, these standard key sizes and standard message length are the following ones:\n")

print("a: 1024 bits.")

print("b: 2048 bits.")

print("c: 3072 bits.")

print("d: 4096 bits.")

print("e: 7680 bits.")

print("f: 8192 bits.\n")

print("The standard key sizes and standard message length showed above are used by the RSA asymmetric encryption and decryption algorithm for the generation and creation of the public and private keys, and for the calculations related to the encryption and decryption process, that are immense and colossal prime numbers generated randomly. However, for the purpose of the present Python program, these standard key sizes showed out above are used also for specifying the size of the message that must be first encrypted and then decrypted by this RSA asymmetric encryption and decryption algorithm.\n")

print("The present Python program, in order to demonstrate the effectiveness of the right implementation of that RSA asymmetric encryption and decryption algorithm, needs to know which standard key size and which standard message length will be used by the RSA asymmetric encryption and decryption algorithm, taking into account that you must choose one of the following standard key sizes and standard message lengths:\n")

print("a: 1024 bits.")

print("b: 2048 bits.")

print("c: 3072 bits.")

print("d: 4096 bits.")

print("e: 7680 bits.")

print("f: 8192 bits.\n")

print("For that purpose, please specify and choose a standard message length from the list shown above:")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_message\_length == "a":

user\_response\_message\_length = 128

elif user\_response\_standard\_message\_length == "b":

user\_response\_message\_length = 256

elif user\_response\_standard\_message\_length == "c":

user\_response\_message\_length = 384

elif user\_response\_standard\_message\_length == "d":

user\_response\_message\_length = 512

elif user\_response\_standard\_message\_length == "e":

user\_response\_message\_length = 960

elif user\_response\_standard\_message\_length == "f":

user\_response\_message\_length = 1024

while user\_response\_standard\_message\_length not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

print("For that purpose, please specify and choose a standard key size from the list shown above:")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_key\_size == "a":

user\_response\_key\_size = 128\*8

elif user\_response\_standard\_key\_size == "b":

user\_response\_key\_size = 256\*8

elif user\_response\_standard\_key\_size == "c":

user\_response\_key\_size = 384\*8

elif user\_response\_standard\_key\_size == "d":

user\_response\_key\_size = 512\*8

elif user\_response\_standard\_key\_size == "e":

user\_response\_key\_size = 960\*8

elif user\_response\_standard\_key\_size == "f":

user\_response\_key\_size = 1024\*8

while user\_response\_standard\_key\_size not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

# Checking the consistency between the message length and the key size

while True:

if user\_response\_standard\_message\_length == user\_response\_standard\_key\_size:

break

else:

print("Error: The selected message length must match the selected key size.")

print("Please choose the same option (a, b, c, d, e, or f) for both message length and key size.")

user\_response\_standard\_message\_length = input("Enter your choice for message length (a, b, c, d, e, or f): ")

user\_response\_standard\_key\_size = input("Enter your choice for key size (a, b, c, d, e, or f): ")

if user\_response\_standard\_message\_length == "a":

user\_response\_message\_length = 128

elif user\_response\_standard\_message\_length == "b":

user\_response\_message\_length = 256

elif user\_response\_standard\_message\_length == "c":

user\_response\_message\_length = 384

elif user\_response\_standard\_message\_length == "d":

user\_response\_message\_length = 512

elif user\_response\_standard\_message\_length == "e":

user\_response\_message\_length = 960

elif user\_response\_standard\_message\_length == "f":

user\_response\_message\_length = 1024

if user\_response\_standard\_key\_size == "a":

user\_response\_key\_size = 128\*8

elif user\_response\_standard\_key\_size == "b":

user\_response\_key\_size = 256\*8

elif user\_response\_standard\_key\_size == "c":

user\_response\_key\_size = 384\*8

elif user\_response\_standard\_key\_size == "d":

user\_response\_key\_size = 512\*8

elif user\_response\_standard\_key\_size == "e":

user\_response\_key\_size = 960\*8

elif user\_response\_standard\_key\_size == "f":

user\_response\_key\_size = 1024\*8

if user\_response\_standard\_key\_size == "a":

msg = input("Enter a message that contains maximum 128 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "b":

msg = input("Enter a message that contains maximum 256 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "c":

msg = input("Enter a message that contains maximum 384 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "d":

msg = input("Enter a message that contains maximum 512 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "e":

msg = input("Enter a message that contains maximum 960 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "f":

msg = input("Enter a message that contains maximum 1024 characters, spaces included: ")

while len(msg) > user\_response\_message\_length:

print(f"The message written down has more than {user\_response\_message\_length} characters, including spaces!")

msg = input(f"Please write down a message that contains {user\_response\_message\_length} characters, including spaces:")

print("Generating RSA keys...")

public\_key, private\_key = generate\_keys(user\_response\_key\_size // 4)

print("RSA keys generated successfully.")

message = msg

encoded\_message = encode\_message(message)

encrypted\_message = [encrypt(char, public\_key) for char in encoded\_message]

decrypted\_message = [decrypt(char, private\_key) for char in encrypted\_message]

decoded\_message = decode\_message(decrypted\_message)

print("\nInitial message:")

print(message)

print("\nPublic key:")

print(public\_key)

print("\nPrivate key:")

print(private\_key)

print("\nEncoded message (encrypted by public key):")

print(encrypted\_message)

print("\nDecoded message (decrypted by private key):")

print(decoded\_message)

**Le voici la version 6 du code python implémentant l’algorithme El-Gamal, en version corrigée, améliorée et optimisée :**

# Importing the random library to generate random numbers

import random

# Importing the sys module to increase the recursion limit

import sys # The sys module provides access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter.

# importing the time library to measure the time that takes the RSA python code, to perform key generation, encoding and decoding of messages

import time

# Increase the recursion limit

sys.setrecursionlimit(10\*\*6)

# To find gcd of two numbers

def gcd(a, b):

if a < b:

return gcd(b, a)

elif a % b == 0:

return b

else:

return gcd(b, a % b)

# For key generation i.e. large random number

def gen\_key(q):

key = random.randint(pow(2, 900), q)

while gcd(q, key) != 1:

key = random.randint(pow(2, 900), q)

return key

def power(a, b, c):

x = 1

y = a

while b > 0:

if b % 2 == 0:

x = (x \* y) % c

y = (y \* y) % c

b = b // 2 # Utilisation de la division entière //

return x % c

# For asymmetric encryption

def encryption(msg, q, h, g):

ct = []

k = gen\_key(q)

s = power(h, k, q)

p = power(g, k, q)

for i in range(0, len(msg)):

ct.append(msg[i])

print("g^k used= ", p)

print("g^ak used= ", s)

for i in range(0, len(ct)):

ct[i] = s \* ord(ct[i])

return ct, p

# For decryption

def decryption(ct, p, key, q):

pt = []

h = power(p, key, q)

for i in range(0, len(ct)):

pt.append(chr(int(ct[i] / h)))

return pt

# This part of the Python program that implements effectively the El-Gamal algorithm, displays the presentation of that Python program (that takes charge of the right choosing of the offered options described below, for the standard key sizes)

print("The present Python program will demonstrate the effectiveness of a right implementation of the El-Gamal asymmetric encryption and decryption algorithm. For that purpose, the implementation of that asymmetric algorithm El-Gamal uses a certain standard key sizes and standard message length determined by international organisations around the world, these standard key sizes and standard message length are the following ones:\n")

print("a: 2048 bits.")

print("b: 3072 bits.")

print("c: 4096 bits.")

print("d: 6144 bits.")

print("e: 8192 bits.")

print("f: 16384 bits.\n")

print("The standard key sizes and standard message length showed above are used by the El-Gamal asymmetric encryption and decryption algorithm for the generation and creation of the public and private keys, and for the calculations related to the encryption and decryption process, that are immense and colossal prime numbers generated randomly. However, for the purpose of the present Python program, these standard key sizes showed out above are used also for specifying the size of the message that must be first encrypted and then decrypted by this El-Gamal asymmetric encryption and decryption algorithm.\n")

print("The present Python program, in order to demonstrate the effectiveness of the right implementation of that El-Gamal asymmetric encryption and decryption algorithm, needs to know which standard key size and which standard message length will be used by the El-Gamal asymmetric encryption and decryption algorithm, taking into account that you must choose one of the following standard key sizes and standard message lengths:\n")

print("a: 2048 bits.")

print("b: 3072 bits.")

print("c: 4096 bits.")

print("d: 6144 bits.")

print("e: 8192 bits.")

print("f: 16384 bits.\n")

print("For that purpose, please specify and choose a standard message length from the list shown above:")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_message\_length == "a":

user\_response\_message\_length = 256

elif user\_response\_standard\_message\_length == "b":

user\_response\_message\_length = 384

elif user\_response\_standard\_message\_length == "c":

user\_response\_message\_length = 512

elif user\_response\_standard\_message\_length == "d":

user\_response\_message\_length = 768

elif user\_response\_standard\_message\_length == "e":

user\_response\_message\_length = 1024

elif user\_response\_standard\_message\_length == "f":

user\_response\_message\_length = 2048

while user\_response\_standard\_message\_length not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

print("For that purpose, please specify and choose a standard key size from the list shown above:")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_key\_size == "a":

user\_response\_key\_size = 256\*8

elif user\_response\_standard\_key\_size == "b":

user\_response\_key\_size = 384\*8

elif user\_response\_standard\_key\_size == "c":

user\_response\_key\_size = 512\*8

elif user\_response\_standard\_key\_size == "d":

user\_response\_key\_size = 768\*8

elif user\_response\_standard\_key\_size == "e":

user\_response\_key\_size = 1024\*8

elif user\_response\_standard\_key\_size == "f":

user\_response\_key\_size = 2048\*8

while user\_response\_standard\_key\_size not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

# Checking the consistency between the message length and the key size

while True:

if user\_response\_standard\_message\_length == user\_response\_standard\_key\_size:

break

else:

print("Error: The selected message length must match the selected key size.")

print("Please choose the same option (a, b, c, d, e, or f) for both message length and key size.")

user\_response\_standard\_message\_length = input("Enter your choice for message length (a, b, c, d, e, or f): ")

user\_response\_standard\_key\_size = input("Enter your choice for key size (a, b, c, d, e, or f): ")

if user\_response\_standard\_message\_length == "a":

user\_response\_message\_length = 256

elif user\_response\_standard\_message\_length == "b":

user\_response\_message\_length = 384

elif user\_response\_standard\_message\_length == "c":

user\_response\_message\_length = 512

elif user\_response\_standard\_message\_length == "d":

user\_response\_message\_length = 768

elif user\_response\_standard\_message\_length == "e":

user\_response\_message\_length = 1024

elif user\_response\_standard\_message\_length == "f":

user\_response\_message\_length = 2048

if user\_response\_standard\_key\_size == "a":

user\_response\_key\_size = 256\*8

elif user\_response\_standard\_key\_size == "b":

user\_response\_key\_size = 384\*8

elif user\_response\_standard\_key\_size == "c":

user\_response\_key\_size = 512\*8

elif user\_response\_standard\_key\_size == "d":

user\_response\_key\_size = 768\*8

elif user\_response\_standard\_key\_size == "e":

user\_response\_key\_size = 1024\*8

elif user\_response\_standard\_key\_size == "f":

user\_response\_key\_size = 2048\*8

if user\_response\_standard\_key\_size == "a":

msg = input("Enter a message that contains maximum 256 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "b":

msg = input("Enter a message that contains maximum 384 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "c":

msg = input("Enter a message that contains maximum 512 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "d":

msg = input("Enter a message that contains maximum 768 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "e":

msg = input("Enter a message that contains maximum 1024 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "f":

msg = input("Enter a message that contains maximum 2048 characters, spaces included: ")

while len(msg) > user\_response\_message\_length:

print(f"The message written down has more than {user\_response\_message\_length} characters, including spaces!")

msg = input(f"Please write down a message that contains {user\_response\_message\_length} characters, including spaces:")

if user\_response\_standard\_key\_size == "a":

q=random.randint(pow(2, 900),pow(2, 2048))

elif user\_response\_standard\_key\_size == "b":

q=random.randint(pow(2, 900),pow(2, 3072))

elif user\_response\_standard\_key\_size == "c":

q=random.randint(pow(2, 900),pow(2, 4096))

elif user\_response\_standard\_key\_size == "d":

q=random.randint(pow(2, 900),pow(2, 6144))

elif user\_response\_standard\_key\_size == "e":

q=random.randint(pow(2, 900),pow(2, 8192))

elif user\_response\_standard\_key\_size == "f":

q=random.randint(pow(2, 900),pow(2, 16384))

print("Generating El-Gamal keys...")

g = random.randint(2, q)

key = gen\_key(q)

h = power(g, key, q)

print("El-Gamal keys generated successfully.")

print("\ng used=", g)

print("g^a used=", h)

ct, p = encryption(msg, q, h, g)

print("\nOriginal Message=", msg)

print("\nEncrypted Message=", ct)

pt = decryption(ct, p, key, q)

d\_msg = ''.join(pt)

print("\nDecrypted Message=", d\_msg)

**Le voici la version 5 du code python implémentant l’algorithme ECC, en version corrigée, améliorée et optimisée :**

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives import serialization, hashes

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives.kdf.hkdf import HKDF

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

import ecdsa

import os

# Generate an ECC key pair

def generate\_ecc\_key\_pair(curve):

private\_key = ec.generate\_private\_key(curve, default\_backend())

public\_key = private\_key.public\_key()

return private\_key, public\_key

# Calculate ECC shared secret

def calculate\_shared\_secret(private\_key, public\_key):

shared\_secret = private\_key.exchange(ec.ECDH(), public\_key)

return shared\_secret

# Generate a derived secret key from the shared secret

def derive\_secret\_key(shared\_secret, length):

derived\_key = HKDF(

algorithm=hashes.SHA256(),

length=length,

salt=None,

info=b'ECC Key Derivation',

backend=default\_backend()

).derive(shared\_secret)

return derived\_key

# Encrypt a message with AES-GCM

def encrypt\_message\_AES\_GCM(message, key):

iv = os.urandom(16)

encryptor = Cipher(

algorithms.AES(key),

modes.GCM(iv),

backend=default\_backend()

).encryptor()

ciphertext = encryptor.update(message.encode('utf-8')) + encryptor.finalize()

tag = encryptor.tag

return (ciphertext, iv, tag)

# Decrypt a message with AES-GCM

def decrypt\_message\_AES\_GCM(ciphertext, iv, tag, key):

decryptor = Cipher(

algorithms.AES(key),

modes.GCM(iv, tag),

backend=default\_backend()

).decryptor()

plaintext = decryptor.update(ciphertext) + decryptor.finalize()

return plaintext

# This part of the Python program that implements effectively the ECC algorithm, displays the presentation of that Python program (that takes charge of the right choosing of the offered options described below, for the standard key sizes)

print("The present Python program will demonstrate the effectiveness of a right implementation of the ECC asymmetric encryption and decryption algorithm. For that purpose, the implementation of that asymmetric algorithm ECC uses a certain standard key sizes and standard message length determined by international organisations around the world, these standard key sizes and standard message length are the following ones:\n")

print("a: 256 bits.")

print("b: 384 bits.")

print("c: 409 bits.")

print("d: 512 bits.")

print("e: 521 bits.")

print("f: 571 bits.\n")

print("The standard key sizes and standard message length showed above are used by the ECC asymmetric encryption and decryption algorithm for the generation and creation of the public and private keys, and for the calculations related to the encryption and decryption process, that are immense and colossal prime numbers generated randomly. However, for the purpose of the present Python program, these standard key sizes showed out above are not used for specifying the size of the message that must be first encrypted and then decrypted by this ECC asymmetric encryption and decryption algorithm.\n")

print("The reason of that statement is simple: the ECC assymetric encryption and decryption algorithm only need the key sizes showed above, to encrypt and decrypt either short, medium and long messages, those messages usually have the following standard message lenght:")

print("a: 2048 bits.")

print("b: 3072 bits.")

print("c: 4096 bits.")

print("d: 6144 bits.")

print("e: 8192 bits.")

print("f: 16384 bits.\n")

print("That statement said, the present Python program, in order to demonstrate the effectiveness of the right implementation of that ECC asymmetric encryption and decryption algorithm, needs to know which standard key size and which standard message length will be used by the ECC asymmetric encryption and decryption algorithm, taking into account that you must choose one of the following standard key sizes and standard message lengths showed below:\n")

print("Standard message lenghts:")

print("a: 2048 bits.")

print("b: 3072 bits.")

print("c: 4096 bits.")

print("d: 6144 bits.")

print("e: 8192 bits.")

print("f: 16384 bits.\n")

print("Standard key sizes:")

print("a: 256 bits.")

print("b: 304 bits.")

print("c: 409 bits.")

print("d: 512 bits.")

print("e: 521 bits.")

print("f: 571 bits.\n")

print("For that purpose, please specify and choose a standard message length from the list shown above:")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_message\_length == "a":

user\_response\_message\_length = 256

elif user\_response\_standard\_message\_length == "b":

user\_response\_message\_length = 384

elif user\_response\_standard\_message\_length == "c":

user\_response\_message\_length = 512

elif user\_response\_standard\_message\_length == "d":

user\_response\_message\_length = 768

elif user\_response\_standard\_message\_length == "e":

user\_response\_message\_length = 1024

elif user\_response\_standard\_message\_length == "f":

user\_response\_message\_length = 2048

while user\_response\_standard\_message\_length not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_message\_length = input("Enter your choice (a, b, c, d, e, or f): ")

print("For that purpose, please specify and choose a standard key size from the list shown above:")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_key\_size == "a":

user\_response\_key\_size = 256

elif user\_response\_standard\_key\_size == "b":

user\_response\_key\_size = 384

elif user\_response\_standard\_key\_size == "c":

user\_response\_key\_size = 409

elif user\_response\_standard\_key\_size == "d":

user\_response\_key\_size = 512

elif user\_response\_standard\_key\_size == "e":

user\_response\_key\_size = 521

elif user\_response\_standard\_key\_size == "f":

user\_response\_key\_size = 571

while user\_response\_standard\_key\_size not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_key\_size = input("Enter your choice (a, b, c, d, e, or f): ")

# Checking the consistency between the message length and the key size

while True:

if user\_response\_standard\_message\_length == user\_response\_standard\_key\_size:

break

else:

print("Error: The selected message length must match the selected key size.")

print("Please choose the same option (a, b, c, d, e, or f) for both message length and key size.")

user\_response\_standard\_message\_length = input("Enter your choice for message length (a, b, c, d, e, or f): ")

user\_response\_standard\_key\_size = input("Enter your choice for key size (a, b, c, d, e, or f): ")

if user\_response\_standard\_key\_size == "a":

msg = input("Enter a message that contains maximum 256 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "b":

msg = input("Enter a message that contains maximum 384 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "c":

msg = input("Enter a message that contains maximum 512 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "d":

msg = input("Enter a message that contains maximum 768 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "e":

msg = input("Enter a message that contains maximum 1024 characters, spaces included: ")

elif user\_response\_standard\_key\_size == "f":

msg = input("Enter a message that contains maximum 2048 characters, spaces included: ")

while len(msg) > user\_response\_message\_length:

print(f"The message written down has more than {user\_response\_message\_length} characters, including spaces!")

msg = input(f"Please write down a message that contains {user\_response\_message\_length} characters, including spaces:")

print("Now that a standard key size and a standanrd message lenght has been chosed by the user, now the ECC assymetric encryption and decryption algorithm needs now to know which elliptic curve will be used, to perform all the process associated to this ECC assymetric algorithm, like the creation of public and private keys, the creation of shared keys and the encryption and decryption of messages.")

print("In order to encrypt and decrypt clear and plain messages, the ECC assymetric encryption and decryption algorithm, implemented by the present program, needs to take one of the following elliptic curves, among an immense range of options determined by international organisations:")

print("a: SECP256R1.")

print("b: SECP384R1.")

print("c: SECT409R1.")

print("d: BrainpooolP512R1.")

print("e: SECP521R1.")

print("f: SECT571K1.\n")

print("For that purpose, please specify and choose an elliptic curve using the standard key size chosen above, from the list shown above:")

user\_response\_standard\_elliptic\_curve = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_elliptic\_curve == "a":

user\_response\_elliptic\_curve = ec.SECP256R1()

elif user\_response\_standard\_elliptic\_curve == "b":

user\_response\_elliptic\_curve = ec.SECP384R1()

elif user\_response\_standard\_elliptic\_curve == "c":

user\_response\_elliptic\_curve = ec.SECT409R1()

elif user\_response\_standard\_elliptic\_curve == "d":

user\_response\_elliptic\_curve = ec.BrainpoolP512R1()

elif user\_response\_standard\_elliptic\_curve == "e":

user\_response\_elliptic\_curve = ec.SECP521R1()

elif user\_response\_standard\_elliptic\_curve == "f":

user\_response\_elliptic\_curve = ec.SECT571K1()

while user\_response\_standard\_elliptic\_curve not in {"a", "b", "c", "d", "e", "f"}:

print("Invalid option. Please choose a valid option (a, b, c, d, e, or f).")

user\_response\_standard\_elliptic\_curve = input("Enter your choice (a, b, c, d, e, or f): ")

if user\_response\_standard\_elliptic\_curve == "a":

user\_response\_elliptic\_curve = ec.SECP256R1()

elif user\_response\_standard\_elliptic\_curve == "b":

user\_response\_elliptic\_curve = ec.SECP384R1()

elif user\_response\_standard\_elliptic\_curve == "c":

user\_response\_elliptic\_curve = ec.SECT409R1()

elif user\_response\_standard\_elliptic\_curve == "d":

user\_response\_elliptic\_curve = ec.BrainpoolP512R1()

elif user\_response\_standard\_elliptic\_curve == "e":

user\_response\_elliptic\_curve = ec.SECP521R1()

elif user\_response\_standard\_elliptic\_curve == "f":

user\_response\_elliptic\_curve = ec.SECT571K1()

# Checking the consistency between the key size and the elliptic curve to chose

while True:

if user\_response\_standard\_elliptic\_curve == user\_response\_standard\_key\_size:

break

else:

print("Error: The selected kwy size must match the selected elliptic curve.")

print("Please choose the same option (a, b, c, d, e, or f) for both key size and elliptic curve.")

user\_response\_standard\_elliptic\_curve = input("Enter your choice for standard elliptic curve (a, b, c, d, e, or f): ")

user\_response\_standard\_key\_size = input("Enter your choice for key size (a, b, c, d, e, or f): ")

# Generate an ECC key pair

private\_key, public\_key = generate\_ecc\_key\_pair(user\_response\_elliptic\_curve)

# Display information

print("\nClear Message :")

print(msg)

print("\nECC Private Key:")

print(private\_key.private\_numbers().private\_value)

print("\nECC Public Key :")

print(public\_key.public\_numbers().x)

# Calculate the shared secret

shared\_secret = calculate\_shared\_secret(private\_key, public\_key)

# Derivation of the secret key

derived\_key = derive\_secret\_key(shared\_secret, 32) # Clé AES 256 bits

print("\nCiphertext pubKey:")

print(derived\_key.hex())

# Encryption key

print("\nEncryption Key:")

print(derived\_key.hex())

# Decryption key

print("\nDecryption Key:")

print(derived\_key.hex())

# Encrypt the message with AES-GCM

ciphertext, iv, tag = encrypt\_message\_AES\_GCM(msg, derived\_key)

# Display the encrypted message

print("\nEncrypted Message:")

print({

'ciphertext': ciphertext.hex(),

'nonce': iv.hex(),

'authTag': tag.hex(),

'ciphertextPubKey': (public\_key.public\_numbers().x, public\_key.public\_numbers().y)

})

# Decrypt the message with AES-GCM

decrypted\_message = decrypt\_message\_AES\_GCM(ciphertext, iv, tag, derived\_key)

# Display the decrypted message

print("\nDecrypted Message:")

print(decrypted\_message.decode('utf-8'))