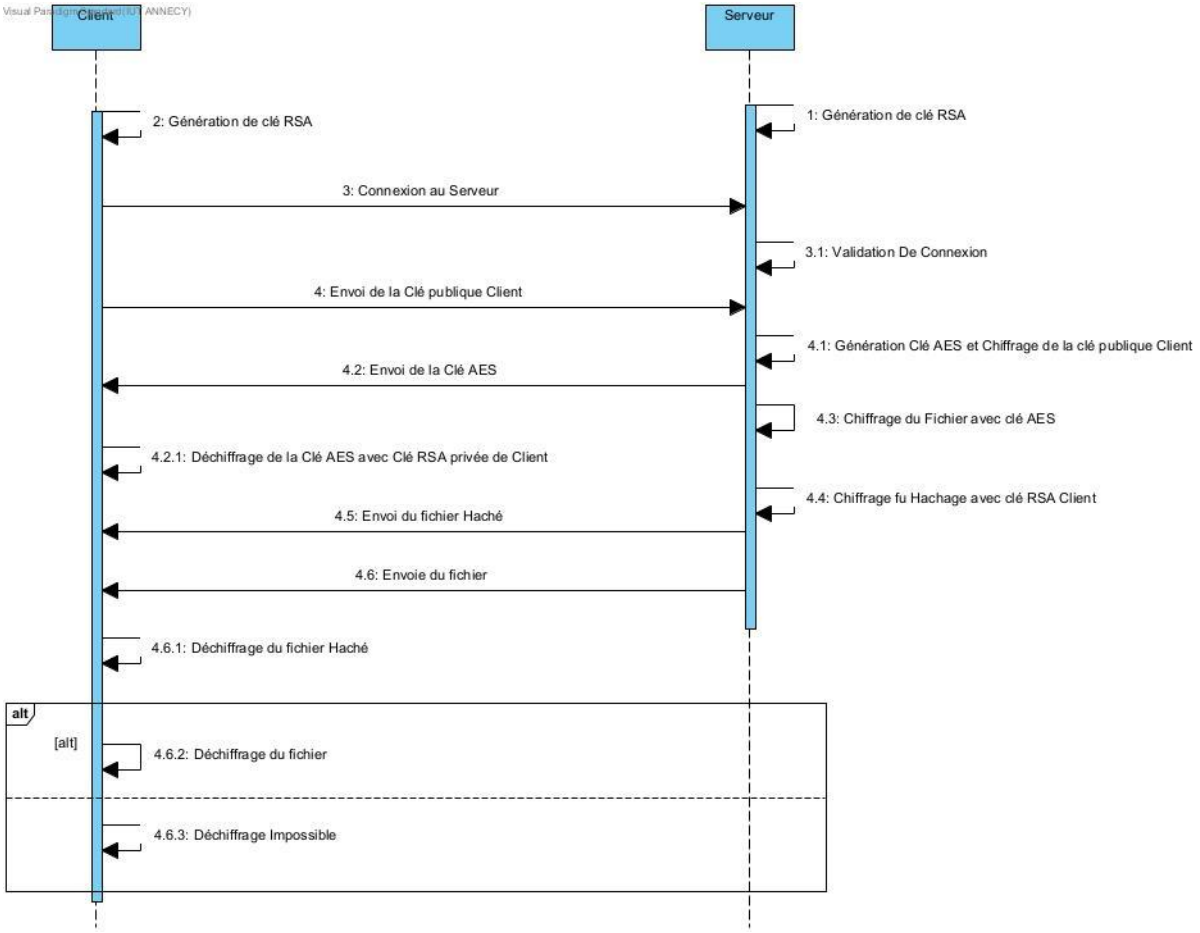
RAPPORT DE CRYPTOGRAPHIE

MIHOUBI Marouane

DIAGRAMME DE SEQUENCE :



RAPPORT DE CRYPTOGRAPHIE

MIHOUBI Marouane

EXPLICATION DU FONCTIONNEMENT DU CODE :

Le serveur commence par initialiser une connexion et génère une paire de clés RSA pour sécuriser les échanges avec le client.

```
server = Server(5000)
(pubkeyS, privkeyS) = server.generateKeys()
```

Dès que le client se connecte, il envoie sa clé publique au serveur.

```
server.waitForConnection()

client_publickey = server.receiveMessage()
client_publickey = PublicKey.load_pkcs1(client_publickey)
```

Parallèlement, le serveur chiffre un message de test avec la clé publique RSA du client pour avoir une communication sécurisée. Une fois chiffré, le message est envoyé au client.

```
encrypt_server = rsa.encrypt('test'.encode('utf8'), client_publickey)
server.sendMessage(encrypt_server)
```

À ce stade, le client reçoit le message chiffré du serveur et le déchiffre avec sa propre clé privée RSA, prouvant ainsi la réception et le déchiffrement sécurisé du message.

```
hashed_server=client.receiveMessage()
message_decrypt = rsa.decrypt(rcv_encrypted, privkeyC)

message=str(message_decrypt.decode("utf-8"))
```

Simultanément, le serveur chiffre le fichier avec AES, en utilisant un mot de passe dérivé du chiffrement RSA. Une fois chiffré, le fichier est envoyé au client.

```
server.encryptAES(f, "test")
f2 = "input/test.txt.aes"
server.sendFile(filename=f2)
```

De son côté, le client reçoit le fichier chiffré du serveur et commence par le déchiffrer avec AES en utilisant le mot de passe préalablement partagé entre le client et le serveur.

```
bfile = client.receiveFile()
f = "output/test.txt.aes"

client.saveFile(bytes=bfile, filename=f)
client.decryptAES(f, message)
```

Après le déchiffrement, le client calcule le hash du fichier déchiffré et le compare au hash reçu du serveur pour vérifier l'intégrité des données.

```
file_content=client.read_file_content("output/solved.txt")
hashed_client=client.compute_hash(file_content)
```

En fonction du résultat de la vérification du hash, le client affiche si la vérification est conforme ou non. Enfin, une fois toutes les opérations terminées, le client ferme la connexion avec le serveur.

RAPPORT DE CRYPTOGRAPHIE

MIHOUBI Marouane

IMPLEMENTATION ET UTILISATION D'AES ET RSA :

RSA : Utilisé pour l'échange de clés publiques entre le client et le serveur, assurant la confidentialité de la communication. RSA utilise le chiffrement asymétrique, où les clés publiques sont utilisées pour chiffrer et les clés privées pour déchiffrer. L'implémentation de RSA utilise la bibliothèque RSA pour la génération de clés RSA ainsi que pour le chiffrement et le déchiffrement des messages de clé.

AES : Utilisé pour chiffrer le fichier échangé entre le client et le serveur. AES utilise le chiffrement symétrique pour la confidentialité des données. Une clé AES est dérivée à partir d'un mot de passe partagé entre le client et le serveur. L'implémentation d'AES utilise la bibliothèque Cryptography pour la gestion des opérations de chiffrement et de déchiffrement. Le fichier est chiffré en utilisant une clé dérivée d'un mot de passe fourni par l'utilisateur. Le schéma de rembourrage PKCS7 est utilisé pour garantir que la taille des données est compatible avec les exigences de l'algorithme AES.

CHOIX DE LA TAILLE :

Pour AES : La clé est dérivée d'un mot de passe fourni par l'utilisateur en utilisant la fonction de hachage SHA-256. La taille de la clé AES est de 256 bits (correspondant à la longueur du hachage SHA-256).

```python
def encryptAES(self, nomFichier, mdp):
    with open(nomFichier, 'rb') as f:
        file_content = f.read()

    key = hashlib.sha256(mdp.encode()).digest()
```

Pour RSA : Les clés RSA sont générées avec une taille de 512 bits. Cette taille peut être ajustée pour répondre à des exigences de sécurité plus strictes.

```python
def generateKeys(self):
    return rsa.newkeys(512)
```

CHOIX DES LIBRAIRIES :

```python
import socket
import rsa
from rsa.key import *
import hashlib
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
```

Cryptography: Choisie pour sa robustesse et sa compatibilité avec les normes de sécurité modernes. Elle offre des fonctionnalités complètes de chiffrement et de hachage.

RSA : Sélectionnée pour sa facilité d'utilisation et sa compatibilité avec Python pour l'implémentation de RSA. Elle offre des fonctionnalités complètes pour la génération de clés et le chiffrement RSA.