

Master 1 Cryptis Mathématiques

Réseaux et Système

Rapport projet TP

Conception d'un protocole de communication basé TCP sécurisé par RSA

Réalisé par :
Fatma KOUIDER

Année Universitaire : **2018-2019**

Table des matières

1	Protocole de sécurité RSA	2
1.1	Outils de sécurité :	2
1.1.1	Génération de la clé publique :	2
1.1.2	Génération de la clé privée :	2
1.1.3	Chiffrement/Déchiffrement :	2
1.2	Utilisation des outils de sécurité :	3
1.2.1	Génération de la clé publique :	3
1.2.2	Génération de la clé privée :	3
2	Protocole TCP	4
2.1	Serveur TCP :	4
2.1.1	Communication : "Client → Serveur"	4
2.1.2	Communication : "Serveur → Client"	5
2.2	Client TCP :	6
2.2.1	Communication : "Client → Serveur"	6
2.2.2	Communication : "Serveur → Client"	7
3	Affichage	8
3.1	Console Serveur	8
3.2	Console Client	8
3.3	Communication : "Client → Serveur"	9
3.4	Communication : "Serveur → Client"	10
4	Conclusion	10

1 Protocole de sécurité RSA

Dans cette partie je vais expliquer comment j'ai procédé pour effectuer le chiffrement/déchiffrement des messages échangés entre le Client/Serveur.

1.1 Outils de sécurité :

Pour effectuer le chiffrement/déchiffrement des messages échangés j'ai besoin de définir certaines fonctions relatives au protocole de sécurité RSA.

Il faut noter que cette partie est commune aux programmes sources Python du client et du serveur.

Etant donné que l'exposant public e_A du Client est égale à e_B l'exposant public du Serveur, je vais noter $e_A = e_B = 65537 = E$.

1.1.1 Génération de la clé publique :

J'ai défini la fonction `GeneratePrimeNumber()` qui retourne deux nombres premiers compris dans un intervalle que j'ai précisé ([257,1000]). Le choix de 257 vient du fait que E doit être inférieur au module, en ce qui concerne 1000 je peux aller plus loin.

Cette fonction me permet de choisir deux nombres premiers P et Q tels que $P \times Q = N$ est le module. D'où la clé publique : (N, E) .

```
def GeneratePrimeNumber():
    L = [1 for i in range(257,1000)]
    while 1:
        #Choisir un nombre "num" aléatoirement depuis la liste L
        num = random.choice(L)

        #Vérifier si le nombre "num" est premier en utilisant la fonction "openssl"
        commande = "openssl prime "
        r = subprocess.run(commande+str(num), shell=True, stdout=subprocess.PIPE)
        resultat_openssl = r.stdout

        #Utiliser l'expression régulière ("is prime")
        regexp = re.compile(r'is prime')

        #Chercher la chaîne de caractère "is prime" dans la chaîne de caractère "resultat_openssl"
        #Si "is prime" est dans "resultat_openssl" on sort de la boucle "while"
        if regexp.search(str(resultat_openssl)):
            break
    return num
```

FIGURE 1 – Génération de nombres premiers

1.1.2 Génération de la clé privée :

Afin de générer la clé privée j'ai défini deux fonctions :

- La fonction $egcd(a, b)$: calcule par l'algorithme d'Euclide étendue le pgcd de E et $\Phi(N) = (P-1)(Q-1)$ et D l'inverse de E modulo $\Phi(N)$.
- La fonction $modinv(a, m)$: Vérifie que j'ai bien $pgcd(E, \Phi(N)) = 1$ ensuite calcule D modulo $\Phi(N)$ afin d'avoir une valeur positive.

D'où la clé privée : (N, D) .

```
def egcd(a, b):
    x, y, u, v = 0, 1, 1, 0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b, a, x, y, u, v = a, r, u, v, m, n
    gcd = b
    return gcd, x, y

#Définir une fonction qui vérifie que l'inverse modulaire existe et calcule cette inverse modulo Phi(Ns)
def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None
    return x % m
```

FIGURE 2 – Inverse modulaire

1.1.3 Chiffrement/Déchiffrement :

Pour effectuer les opérations de chiffrement/déchiffrement j'ai défini la fonction $lpowmod(x, y, n)$ qui calcule les puissances modulaire selon l'algorithme d'exponentiation rapide.

- Chiffrement : La fonction $lpowmod(x, y, n)$ prend en entrée la clé publique (N, E) et m le message à chiffrer et calcule $c = m^E \bmod N$ et le retourne.

- Déchiffrement : La fonction $lpowmod(x, y, n)$ prend en entrée la clé privée (N, D) et c le message à déchiffrer et calcule $m = c^D \bmod N$ et le retourne.

```
def lpowmod(x, y, n):
    """puissance modulaire: (x**y)%n avec x, y et n entiers"""
    result = 1
    while y > 0:
        if y & 1 > 0:
            result = (result * x) % n
        y >>= 1
        x = (x * x) % n
    return result
```

FIGURE 3 – Exponentiation Rapide

1.2 Utilisation des outils de sécurité :

J'ai fait appel aux fonctions définies précédemment dans les deux programmes Client/Serveur comme suit :

1.2.1 Génération de la clé publique :

1. Je vais tout d'abord faire appel à la fonction `GeneratePrimeNumber()` pour générer P et Q .

```
#Générer Ps et Qs
Ps = GeneratePrimeNumber()
while 1:
    Qs = GeneratePrimeNumber()
    if (Ps != Qs):
        break
```

FIGURE 4 – Génération de Ps et Qs - Serveur

```
#Générer Pc et Qc distincts
Pc = GeneratePrimeNumber()
while 1:
    Qc = GeneratePrimeNumber()
    if (Pc != Qc):
        break
```

FIGURE 5 – Génération de Pc et Qc - Client

2. Je vais ensuite calculer le produit de P et Q pour obtenir le module N .

```
#Calcule du module de Serveur Ns = Ps*Qs
Ns = int(Ps) * int(Qs)
```

FIGURE 6 – Module - Serveur

```
#Calcule du module du Client Nc = Pc*Qc
Nc = int(Pc) * int(Qc)
```

FIGURE 7 – Module - Client

1.2.2 Génération de la clé privée :

1. Je vais tout d'abord faire calculer $\Phi(N) = (P - 1)(Q - 1)$.

```
#Calcule de Phi(Ns) = (Ps-1)(Qs-1)
Phi_Ns = (int(Ps)-1) * (int(Qs)-1)
```

FIGURE 8 – $\Phi(N_s)$ - Serveur

```
#Calcule de Phi(Nc) = (Pc-1)(Qc-1)
Phi_Nc = (int(Pc)-1) * (int(Qc)-1)
```

FIGURE 9 – $\Phi(N_c)$ - Client

2. Je vais ensuite faire appel à la fonction $modinv(E, \Phi(N))$ qui va retourner D l'inverse de E modulo $\Phi(N)$.

```
#Calcule de l'exposant secret du Serveur Ds = E^(-1) mod Phi(Ns)
Ds = modinv(E, Phi_Ns)
```

FIGURE 10 – Clé privée Ds - Serveur

```
#Calcule de l'exposant secret Dc = E^(-1) mod Phi(Nc)
Dc = modinv(E, Phi_Nc)
```

FIGURE 11 – Clé privée Dc - Client

2 Protocole TCP

Dans cette partie Je vais illustrer comment j'ai établi une communication Client/Serveur TCP sécurisée

2.1 Serveur TCP :

1. Création d'une socket pour permettre la communication TCP

```
#Création d'une socket pour permettre la communication TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

FIGURE 12 – Création d'une socket

2. Indiquer le numéro de port sur le quel la communication sera effectuée

```
#Communication sur le port numéro 8790
s.bind(('',8790))
```

FIGURE 13 – Indication du numéro de port

3. Attendre une communication Client

```
#Attendre une connexion Client
s.listen(1)
```

FIGURE 14 – Attendre une communication Client

4. Accepter de communiquer avec le Client

```
#Accepter de communiquer avec le Client
connexion, tsap_client = s.accept()
print(tsap_client)
```

FIGURE 15 – Autorisation de connexion

Schéma de communication :

Je vais expliciter le fonctionnement du protocole Serveur TCP en chiffrant les messages échangés avec le Client.

2.1.1 Communication : "Client → Serveur"

1. Lecture du message saisie par le Client.

```
#Lecture du message du Client
ligne = connexion.recv(1024)
```

FIGURE 16 – Lecture du message Client

2. Décoder le message du Client d'une chaîne d'octets à une chaîne en UTF-8 avec la méthode decode()

```
#Passer d'une chaîne d'octets à une chaîne en UTF-8 avec la méthode decode()
ligne = ligne.decode('utf-8')
```

FIGURE 17 – Décodage d'une chaîne d'octets en une chaîne UTF-8

3. Récupérer le message du Client sous forme d'une liste de chiffré dont on enlèvera le caractère ',' avec la méthode `split()`.

```
#Récupérer le message du Client sous forme d'une liste de chiffré dont on enlèvera le caractère ',' avec la méthode split()
msg = str(ligne).split(',')
```

FIGURE 18 – Transformation d'une chaîne de caractère en une liste

4. Parcourir la liste "msg" et déchiffrer chaque élément à l'aide de la fonction "lpowmod" et la clé privée "Ds" du Serveur

```
#Parcourir la liste "msg" et déchiffrer chaque élément à l'aide de la fonction "lpowmod" et la clé privée Ds du Serveur
for ch in msg:
    dechiffrement += chr(lpowmod(int(ch),Ds,Ns))
```

FIGURE 19 – Déchiffrement message

5. Afficher le message du Client après déchiffrement

```
#Afficher le message du Client après déchiffrement
print("Client: ",dechiffrement)
```

FIGURE 20 – Afficher le message déchiffré

2.1.2 Communication : "Serveur → Client"

1. Récupérer la clé publique Nc du Client sous forme d'une chaîne d'octets que nous allons décoder en une chaîne UTF-8 avec la méthode `decode()`.

```
#Récupérer la clé publique Nc du Client
Nc = connexion.recv(1024)
Nc = NA.decode("utf-8")
```

FIGURE 21 – Récupérer la clé publique "Nc" du Client

2. Saisie du message du Serveur.

```
#Saisie du message du Serveur
saisie = input('->')
```

FIGURE 22 – Saisie du message du Serveur

3. Parcourir la chaîne de caractères qui contient le message du Serveur et chiffrer chaque caractère à l'aide de la fonction "lpowmod" et la clé publique du Client

```
#Parcourir la chaîne de caractère qui contient le message du Serveur
#Et chiffrer chaque caractère à l'aide de la fonction "lpowmod" et la clé publique du Client
for ch in saisie:
    chiffrement.append(lpowmod(ord(ch),E,int(Nc)))
```

FIGURE 23 – Chiffrement du message

4. Affecter le résultat dans une chaîne de caractères

```
#Affecter le résultat dans une chaîne de caractère
msg = str(chiffrement).strip('[]')
```

FIGURE 24 – Affecter le chiffré dans une chaîne de caractères

5. Transmettre le message chiffré du Serveur au Client

```
#Transmettre le message chiffré du Serveur au Client
connexion.sendall(bytes(msg,'utf-8'))
```

FIGURE 25 – Transmettre le message chiffré au Client

2.2 Client TCP :

Tout d'abord j'ai demandé au Serveur d'autoriser le Client à communiquer avec le Serveur.

```
#Etablir une connexion avec le Serveur
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tsap_serveur = ('127.0.0.1',8790)
s.connect(tsap_serveur)
```

FIGURE 26 – Connexion avec le Serveur

Ensuite j'ai récupéré le module publique "Ns" du Serveur depuis le fichier "Annuaire.txt".

```
#Récupérer le module publique Ns du Serveur depuis le fichier "Annuaire.txt"
file = open("Annuaire.txt","r")
Ns = file.read()
file.close
```

FIGURE 27 – Récupérer le module "Ns" du Serveur

Schéma de communication :

Je vais expliciter le fonctionnement du protocole Client TCP en sécurisant les messages échangés avec le Serveur.

2.2.1 Communication : "Client → Serveur"

1. Saisie du message du Client

```
#Saisie du message du Client
entree_clavier = input('**')
if not entree_clavier:
    break
```

FIGURE 28 – Saisie message - Client

2. Parcourir la chaîne de caractères qui contient le message du Client et effectuer le chiffrement par la clé publique (Ns,E) du Serveur en utilisant la fonction "lpowmod"

```
for ch in entree_clavier:
    chiffrement.append(lpowmod(ord(ch),E,int(Ns)))
```

FIGURE 29 – Chiffrement message - Client

3. Affecter le résultat du chiffrement dans la chaîne de caractère "msg"

```
#Affecter le résultat du chiffrement dans la chaîne de caractère "msg"
msg = str(chiffrement).strip('[]')
```

FIGURE 30 – Transformation du chiffré en chaîne de caractères - Client

4. Transmettre le message chiffré du Client au Serveur

```
#Transmettre le message chiffré du Client au Serveur
s.sendall(bytes(msg,'utf-8'))
```

FIGURE 31 – Envoie du chiffré au Serveur

2.2.2 Communication : "Serveur → Client"

1. Transmettre le module publique Nc du Client au Serveur

```
#Transmettre le module publique Nc du Client au Serveur
s.sendall(bytes(str(Nc),'utf-8'))
```

FIGURE 32 – Transmission du module publique "Nc" au Serveur

2. Récupérer le message du Serveur

```
#Récupérer le message du Serveur
ligne = s.recv(1024)
if not ligne:
    break
```

FIGURE 33 – Récupérer le message du Serveur

3. Décoder le message du Serveur, d'une chaîne d'octets à une chaîne en UTF-8 par la méthode decode()

```
#Passer d'une chaîne d'octets à une chaîne en UTF-8 avec la méthode decode()
ligne = ligne.decode('utf-8')
```

FIGURE 34 – Décoder une chaîne d'octets en une chaîne UTF-8

4. Récupérer le message décodé du Serveur sous forme d'une liste de chiffré dont on enlèvera le caractère ',' avec la méthode split()

```
#Récupérer le message du Serveur sous forme d'une liste de chiffré dont on enlèvera le caractère ',' avec la méthode split()
msg = str(ligne).split(',')
```

FIGURE 35 – Récupérer le message du Serveur

5. Parcourir la liste "msg" et déchiffrer chaque élément à l'aide de la fonction "lpowmod" et la clé privé Dc du Client

```
#Parcourir la liste "msg" et déchiffrer chaque élément à l'aide de la fonction "lpowmod" et la clé privé Dc du Client
for ch in msg:
    dechiffrement += chr(lpowmod(int(ch),Dc,Nc))
```

FIGURE 36 – Déchiffrement message - Client

6. Afficher le message du Serveur après déchiffrement

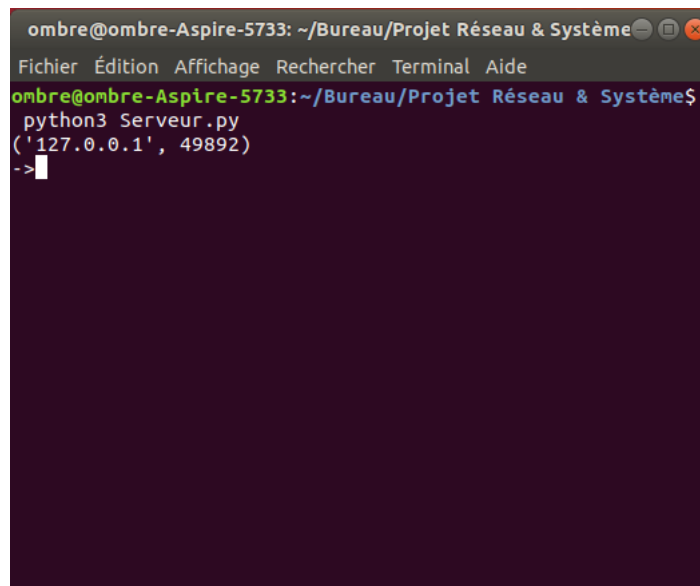
```
#Afficher le message du Serveur après déchiffrement
print("Serveur: ",dechiffrement)
```

FIGURE 37 – Afficher le message déchiffré

3 Affichage

Dans cette partie vous trouverez des prises d'écrans de la communication Client/Serveur de notre projet.

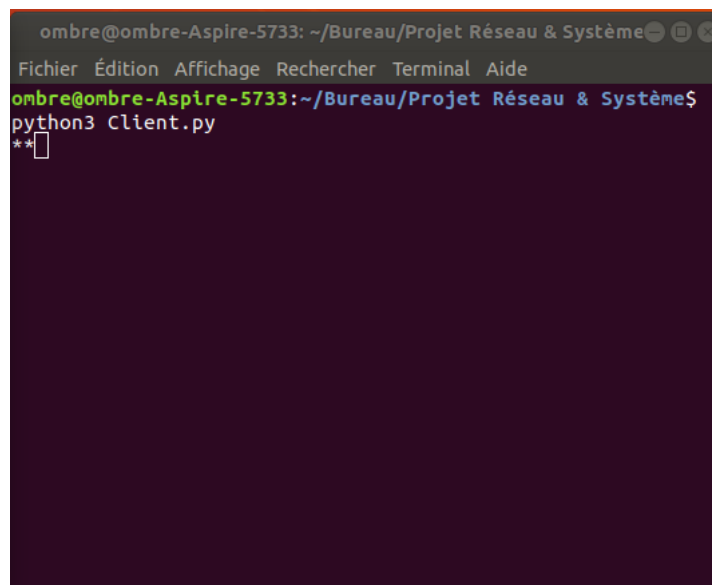
3.1 Console Serveur

A screenshot of a terminal window titled 'ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système'. The window has a menu bar with 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'. The terminal shows the command 'python3 Serveur.py' being executed, which outputs '(\127.0.0.1', 49892)' and a prompt '->'.

```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$
python3 Serveur.py
(\127.0.0.1', 49892)
->
```

FIGURE 38 – Console Serveur

3.2 Console Client

A screenshot of a terminal window titled 'ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système'. The window has a menu bar with 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'. The terminal shows the command 'python3 Client.py' being executed, which outputs '**' and a prompt.

```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$
python3 Client.py
**
```

FIGURE 39 – Console Client

3.3 Communication : "Client → Serveur"

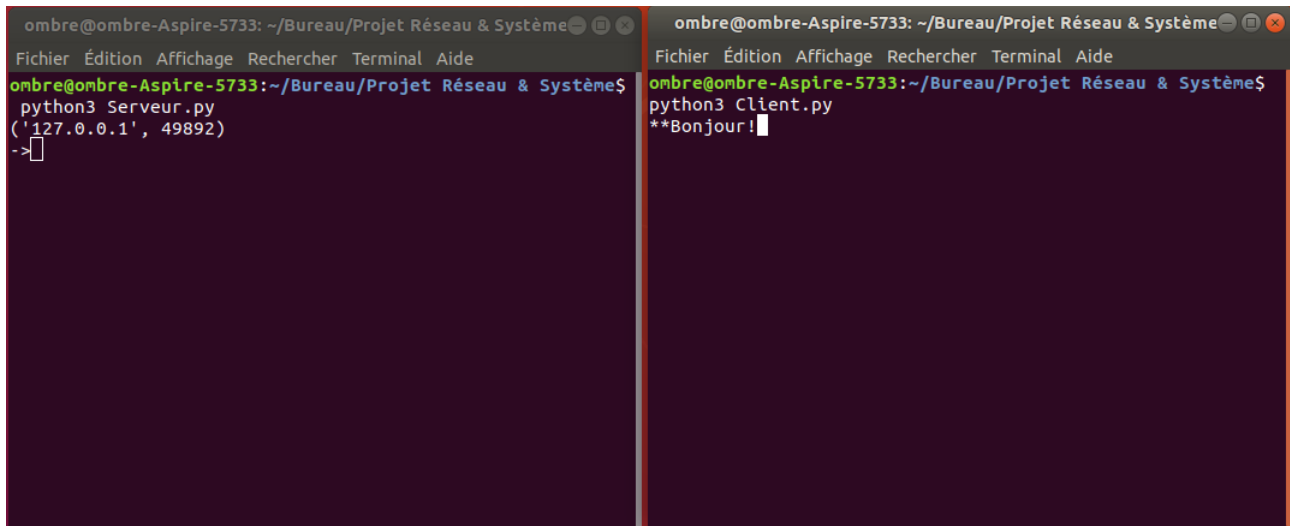


FIGURE 40 – Communication Client → Serveur (Avant envoi)

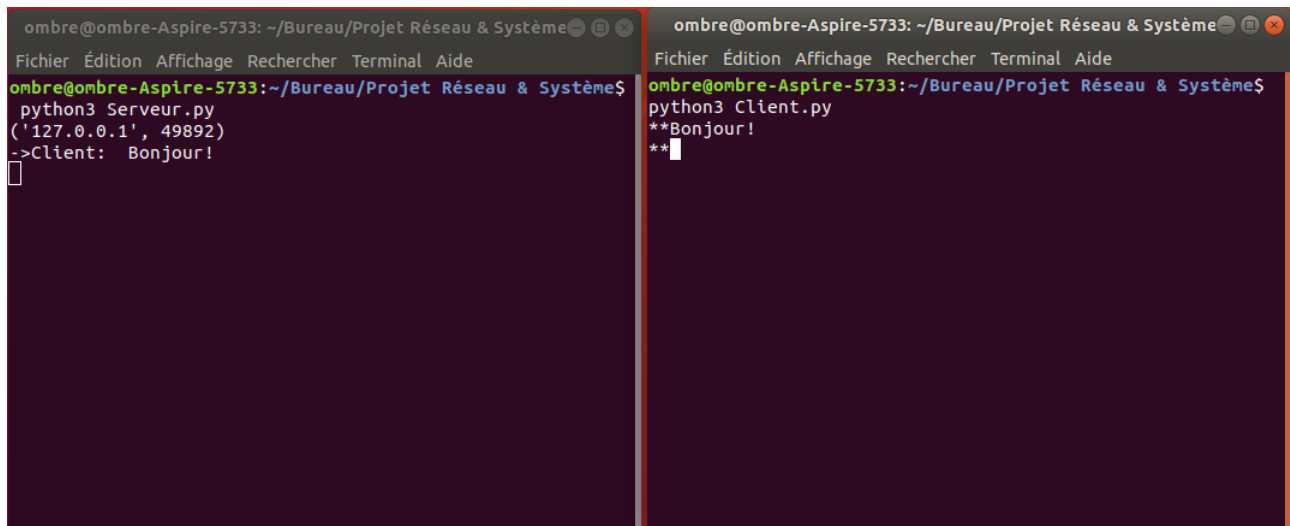
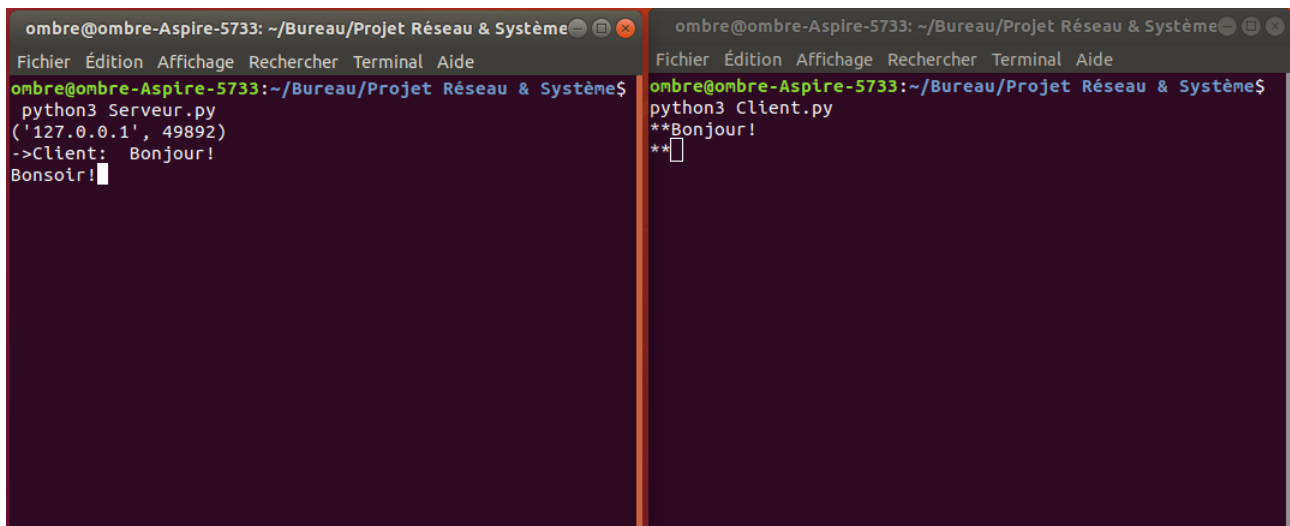


FIGURE 41 – Communication Client → Serveur (Après envoi)

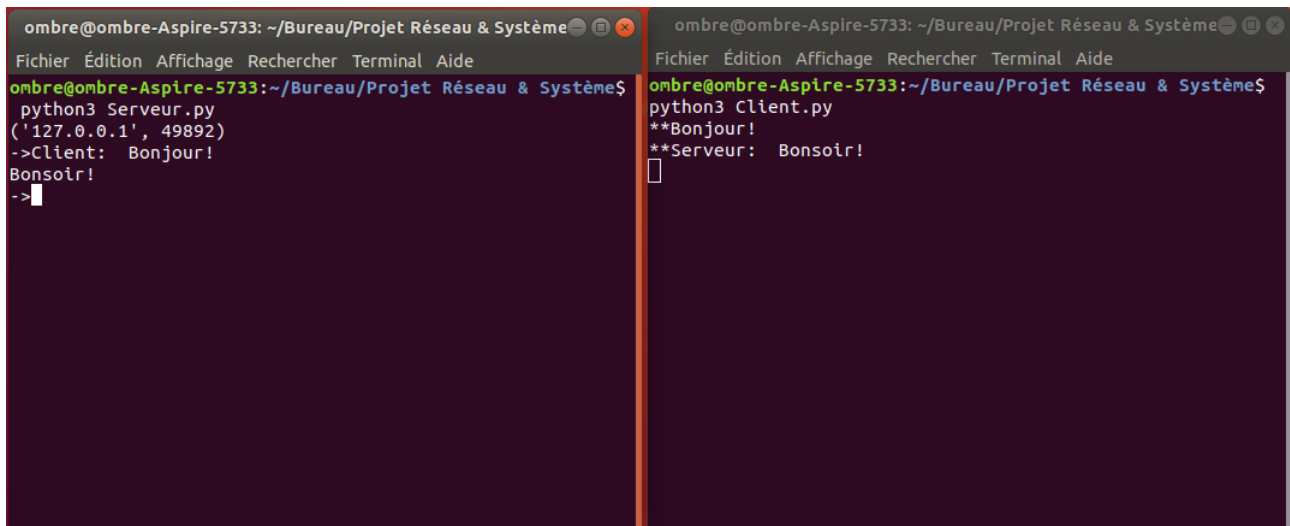
3.4 Communication : "Serveur → Client"



```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$ python3 Serveur.py ('127.0.0.1', 49892)
->Client: Bonjour!
Bonsoir!
```

```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$ python3 Client.py
**Bonjour!
**
```

FIGURE 42 – Communication Serveur → Client (Avant envoi)



```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$ python3 Serveur.py ('127.0.0.1', 49892)
->Client: Bonjour!
Bonsoir!
->
```

```
ombre@ombre-Aspire-5733: ~/Bureau/Projet Réseau & Système
Fichier Édition Affichage Rechercher Terminal Aide
ombre@ombre-Aspire-5733:~/Bureau/Projet Réseau & Système$ python3 Client.py
**Bonjour!
**Serveur: Bonsoir!

```

FIGURE 43 – Communication Serveur → Client (Après envoi)

4 Conclusion

A travers ce projet j'ai appris à sécuriser les communications établies sous le protocole Serveur/Client TCP à l'aide du cryptosystème RSA.