



**Master 1 Maths Cryptis**  
**Projet : Réseaux et systèmes**

**sujet:**

*Conception de protocole de communication basé TCP sécurisé  
par RSA*

**Réalisé par:**

Bahadou Ibtissam  
Boudfor Hajar

## **Table des matières**

1. Introduction
2. choix d'implémentation
3. chiffrement RSA
4. Génération De nombres Premiers de grande taille
- 5 - Génération des exposants et module RSA
6. Protocole de communication serveur-client :
7. Problèmes rencontrés pour ce projet
8. Conclusion
9. Bibliographie

## 1.Introduction :

Python est un langage de programmation qui a une syntaxe claire et simple, ce qui facilite sa compréhension et rend sa programmation plus aisée. Sa syntaxe ne nécessite pas de déclarer le type de variable, ce qui en fait un langage de programmation idéal pour les débutants. De plus, Python est agile et possède de nombreuses bibliothèques utiles, ce qui permet de créer facilement des applications de programmation réseau sécurisées grâce à un chiffrement choisi. En comparant Python à d'autres langages de programmation comme le langage C, on a remarqué que Python est particulièrement adapté à ce type de projet. L'objectif de notre projet est donc de mettre en œuvre ces qualités pour réaliser une application de programmation réseau sécurisée.

## 2. les choix d'implémentations :

**bibliothèques ou Framework :**

```
1  #!/usr/bin/python3
2  import random
3  import subprocess
4  import re
5  import os, socket
```

**Socket** est une bibliothèque Python qui fournit des outils pour la création de programmes de réseau. Elle permet de créer des sockets, qui sont des points finaux pour la communication de données sur un réseau. Cette dernière est particulièrement utile pour la conception de programmes de réseau, tels que des serveurs ou des protocoles de communication. Elle offre également un certain nombre de fonctionnalités avancées, telles que la gestion des adresses IP et des ports, la gestion de la transmission de données en continu et la gestion de la connexion et de la déconnexion de sockets.

**re** fournit des outils pour la manipulation et l'analyse de chaînes de caractères en utilisant des expressions régulières. Les expressions régulières sont des modèles de chaînes de caractères qui peuvent être utilisés pour rechercher ou remplacer des sous-chaînes de caractères dans une chaîne de caractères. La bibliothèque re offre une variété de fonctionnalités pour travailler avec des expressions régulières, telles que la recherche de chaînes de caractères qui correspondent à un modèle donné, le remplacement de sous-chaînes de caractères par une autre chaîne de caractères, et la découpe de chaînes de caractères en groupes en fonction de modèles. La bibliothèque re est particulièrement utile pour l'analyse de données textuelles et pour la validation de données d'entrée.

**Radom** nous a permis de générer des nombres aléatoires et des séquences aléatoires et nous a offert des fonctionnalités pour la génération de nombres entiers aléatoires, de nombres à virgule flottante aléatoires et de nombres complexes aléatoires, ainsi que pour mélanger et sélectionner des éléments d'une liste de manière aléatoire.

**Subprocess** est un module qui permet d'exécuter des processus enfants depuis un programme Python. On peut utiliser **subprocess** pour lancer n'importe quel programme externe depuis Python et pour gérer l'exécution de ces processus. On peut également utiliser **subprocess** pour envoyer des données en entrée à un processus enfant, pour récupérer les données en sortie du processus et pour attendre la fin d'un processus avant de continuer l'exécution de votre programme Python.

### le système d'exploitation:

on a choisi Ubuntu puisqu'il représente la suite du travail qu'on a fait aux séances du TP; En effet, il inclut de nombreux outils de développement, tels que GCC (GNU Compiler Collection) et Python, qu'on a déjà choisi pour la conception de programmes de réseau. Cela signifie qu'on n'a pas installé ces outils séparément et qu'on a immédiatement commencé à travailler sur le protocole de communication.

### 3. Le Chiffrement RSA :

Le chiffrement RSA est un algorithme de cryptographie asymétrique qui utilise deux clefs : une clef publique qui est partagée avec tous et une clef privée qui est gardée secrète. L'algorithme a été inventé en 1978 par Rivest, Shamir et Adleman. La sécurité de ce chiffrement repose sur la difficulté de factoriser des nombres entiers de grande taille qui sont le produit de deux nombres premiers. Plus le nombre est grand, plus la sécurité est grande, il est donc recommandé d'utiliser des nombres d'au moins 1024 bits.

Pour utiliser RSA, il faut d'abord construire les clefs. Pour cela, il faut choisir deux nombres premiers de très grande taille,  $p$  et  $q$ , qui resteront secrets. Leur produit  $n = p * q$  est alors calculé. On calcule également  $\phi(n) = (p-1)(q-1)$  et on choisit un entier  $e$  premier avec  $\phi(n)$ . On calcule ensuite  $d \equiv e^{-1} \bmod \phi(n)$ . On dispose alors de deux clefs :  $(e, n)$  est la clef publique et  $d$  est la clef privée.

Pour envoyer un message chiffré à quelqu'un utilisant RSA, il faut d'abord échanger les clefs publiques. Si Alice veut envoyer un message à Bob, elle chiffre le message avec la clef publique de Bob, en calculant  $c \equiv m^e \bmod n$ . Une fois que Bob reçoit le message, il le déchiffre avec sa clef privée  $d$  en calculant  $m \equiv c^d \bmod n$ . La procédure est inverse si Bob veut envoyer un message à Alice.

### 4. Génération De nombres Premiers de grande taille :

```

13 def GeneratePrimeNumber():
14     random.seed()
15     commande = "openssl prime"
16     n_0 = random.choice('1379')
17     n_i = [random.choice('0123456789') for i in range(0, 11)]
18     n_max = random.choice('123456789')
19     Liste = list(n_max)
20     Liste.extend(n_i)
21     Liste.append(n_0)
22     q = int(''.join(Liste))
23     while True:
24         n_a = random.choice('0123456789')
25         n_b = random.choice('1379')
26         n_i.append(n_a)
27         n_i.append(n_b)
28         p = int(''.join(n_i))
29         r = subprocess.run("{} {}".format(commande, p), shell=True, stdout=subprocess.PIPE)
30         resultat_openssl = r.stdout
31         v = re.compile(r'is prime')
32         p_resultats = v.search(str(resultat_openssl))
33         n_i.pop()
34         n_i.pop()
35         if p_resultats != None:
36             break
37     pfinal = p
38     return(pfinal)

```

Afin de générer un nombre premier, nous avons créé la fonction **GeneratePrimeNumber ()** qui à chaque exécution, produit un nombre premier de 13 chiffres.

"**random.seed()**" initialise le générateur de nombres aléatoires pour qu'il produise des valeurs différentes à chaque exécution du programme .

La fonction **GeneratePrimeNumber ()** entre dans une boucle infinie et génère une liste de 11 chiffres "n\_i" de manière aléatoire à l'aide de la fonction **random. Choice ()**, elle génère également de manière aléatoire deux chiffre n\_a et n\_b pour les ajouter à la liste "n\_i". **P = int (". Join (n\_i))** permet ainsi de concaténer cette liste pour en faire un entier 'p'.

La fonction utilise ensuite les expressions régulières pour vérifier si p est premier en cherchant dans le résultat de "**openssl**" la chaîne de caractères "**is prime**", et en stockant le résultat de cette recherche dans la variable "**p\_resultats**". Si "**p\_resultats**" n'est pas vide, cela signifie que "p" est premier, et la boucle s'arrête. Sinon, la fonction retire les deux derniers chiffres ajoutés à "n\_i" et recommence la boucle.

Une fois la boucle terminée, la fonction retourne le nombre premier généré, stocké dans la variable "**pfinal**".

### Exécution

```
hajar@hajar-T... x hajar@hajar-T... x hajar@hajar-T... x hajar@hajar-T... x
hajar@hajar-ThinkPad-X240:~$ python3 generation.py
7594891406671
5601531842041
hajar@hajar-ThinkPad-X240:~$
```

## 5 - Génération des exposants et module RSA :

### ➤ Algorithme d'Euclide étendu

```
39 def egcd(a, b):
40     x,y, u,v = 0,1, 1,0
41     while a != 0:
42         q, r = b//a, b%a
43         m, n = x-u*q, y-v*q
44         b,a, x,y, u,v = a,r, u,v, m,n
45     gcd = b
46     return gcd, x, y
47
```

### ➤ Inverse modulaire

```
50 def modinv(a, m):
51     gcd, x, y = egcd(a, m)
52     if gcd != 1:
53         return None
54     return x % m
55
```

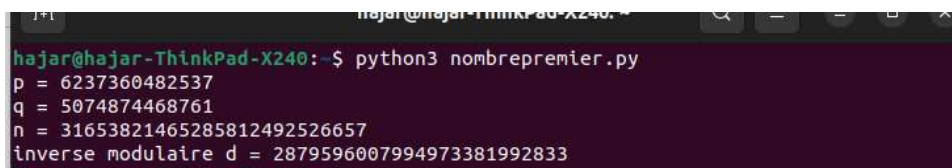
```

59 P = GeneratePrimeNumber()
60 Q = GeneratePrimeNumber()
61 Phi_N = (int(P)-1) * (int(Q)-1)
62 N = int(P)*int(Q)
63 b = modinv(e, Phi_N)
64
65

```

A l'aide de la fonction **GeneratePrimeNumber()** on tire au hasard deux nombres premiers  $p$  et  $q$  et on calcul leurs produit  $n = p * q$  et  $\phi(n) = (p - 1) (q - 1)$ . Connaissant la valeur de l'exposant  $e$ , on calcul l'inverse **modulaire**  $d$  en utilisant la fonction **modinv(e,  $\phi(n)$ )** tout en en respectant la condition de primalité entre  $e$  et l'indicatrice d'Euler qu'on vérifie avec la fonction **egcd(e,  $\phi(n)$ )**. Ces étapes permettent de déterminer la clef secrète d'Alice et de Bob.

### Jeux d'essai :



```

hajar@hajar-ThinkPad-X240:~$ python3 nombrepremier.py
p = 6237360482537
q = 5074874468761
n = 31653821465285812492526657
inverse modulaire d = 2879596007994973381992833

```

## 6. Protocole de communication serveur-client :

Le protocole TCP (Transmission Control Protocol) est utilisé pour établir une connexion entre différents réseaux grâce à l'utilisation d'adresses IP. Il utilise les primitives de connexion suivantes : socket, bind, listen, accept, connect, close, shutdown. Dans ce projet, nous allons utiliser une architecture serveur/client avec TCP.

### ➤ Programmation socket : client et serveur TCP :

Afin d'établir une connexion TCP entre le client et le serveur, le programme d'Alice joue le rôle du client en envoyant une demande de connexion. Le programme de Bob, en tant que serveur, attend cette demande et utilise un numéro de port pour y répondre.

```

10
11 port_number = 8790
12
13
14 #création de socket et attente de connexion de la part du client
15 ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
16 ma_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
17 ma_socket.bind(('', port_number))
18 ma_socket.listen(socket.SOMAXCONN)

```

## Serveur : Bob

```

#create socket ad get port number and server adress
server_adress = socket.gethostname('localhost')
port_number = 8790
ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#connction
try:
    ma_socket.connect((server_adress, port_number))
except Exception as e :
    print("Erreur de connexion",e.args)
    sys.exit(1)

```

## Client : Alice

```

# la fonction qui calcule l'algorithme d'euclide étendu
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

#fonction de L'inverse modulaire L'inverse modulaire
def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None
    return x % m

# Algorithme d'exponentiation rapide
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

P = GeneratePrimeNumber()
Q = GeneratePrimeNumber()

#Calcule du module N = P*Q
n = int(P) * int(Q)

#Calcule de Phi(N) = (P-1)(Q-1)
Phi_n = (int(P)-1) * (int(Q)-1)

# Calcule de L'inverse modulaire
d = modinv(e, Phi_n)

```

Ce programme permet de déterminer les paramètres de chiffrement RSA, notamment les valeurs  $e$ ,  $d$  et  $n$ . Il utilise également l'algorithme d'exponentiation rapide pour calculer  $x^y \bmod n$  grâce à la fonction `lpowmod(x, y, n)`.

Alice et Bob utilisent ainsi ce programme pour pouvoir chiffrer leurs messages respectivement avec la clé publique de Bob et d'Alice.

### ➤ Echange entre client et serveur :

Alice et Bob ont établi une connexion et Alice envoie le premier message chiffré. Bob partage avec Alice  $n_B$ .

Alice reçoit la clé publique  $e_B$  et  $n_B$  et insère son message. Afin de chiffrer son message le programme ci-dessous effectue les étapes suivantes : il découpe le message en mots et les ajoutent dans une liste (**ligne 92**) , convertit chaque mot en hexadécimal en utilisant l'encodage utf-8 (**ligne 93**) , convertit ces mots en entiers et les code en utilisant  $e$  et  $n_B$  (**ligne 94-95**) , puis rassemble tous ces éléments en les séparant par un espace pour obtenir le message chiffré (**ligne 96**)

Enfin Il ajoute la valeur  $n_A$  au message chiffré en le séparant d'un espace (**ligne 97**) et l'envoie à Bob pour qu'il puisse à son tour de chiffrer son propre message .

```

86 N_bob = int(ma_socket.recv(1024))
87 while 1:
88
89     message = input("message?")
90     if message:
91         re_separateur = re.compile(r"['']+")
92         liste = re_separateur.split(str(message))
93         liste_en_hex = [bytes(u,'utf8').hex() for u in liste]
94         conv_to_int = [int(x,16) for x in liste_en_hex]
95         message_chiffré_par_mot = [str(lpowmod(x,e,N_bob)) for x in conv_to_int]
96         message_chiffré = ''.join(message_chiffré_par_mot)
97         message_chiffré_envoyé = message_chiffré + ' ' + str(N_alice)
98         print("Voilà le message chiffré envoyé:",message_chiffré_envoyé)
99         ma_socket.sendall(bytes(message_chiffré_envoyé, 'utf8'))
100
101

```

Exécution :

```

hajar@hajar-ThinkPad-X240: ~
hajar@hajar-ThinkPad-X240: ~
hajar@hajar-ThinkPad-X240:~$ python3 Alice.py
message? bonjour
Voilà le message chiffré envoyé: 19225919137641905542673476 24181091167565979312
993941
message?

```



Bob reçoit le message chiffré d'Alice et sa clé publique eA. Le message chiffré est une chaîne de bytes avec des éléments séparés par des espaces, car le message a été chiffré mot par mot.

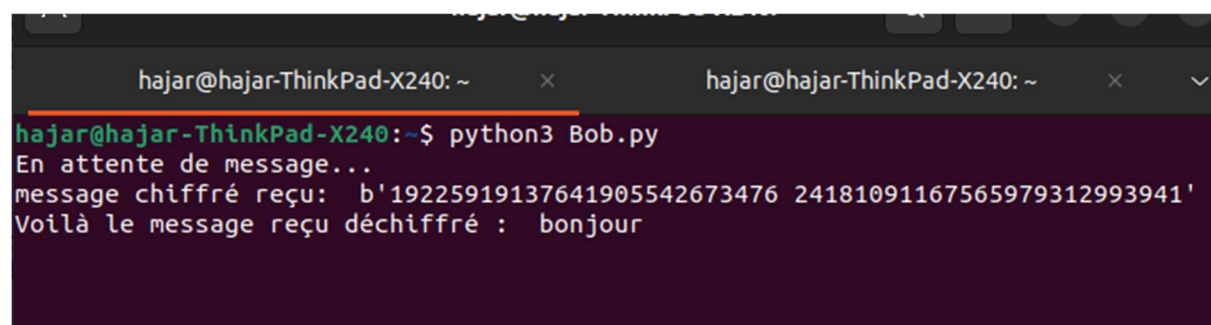
Pour déchiffrer le message d'Alice, le programme ci-dessous effectue les étapes suivantes : il convertit le message en chaîne de caractères utf8 (l.92), place les éléments dans une liste et les convertit en entiers (l.93-94), sépare le message chiffré et la clé publique envoyés par Alice (l.95) (la clé publique d'Alice est le dernier élément de la liste et le reste est le message reçu), puis décode le message avec sa clé privée en utilisant la fonction `lpowmod()` (l.96) . Une fois le message déchiffré, des conversions sont effectuées pour retrouver le message clair tel qu'il a été envoyé par Alice. (l.97-99)

```

84 print("En attente de message...")
85 (nouvelle_connexion, tsap_from) = ma_socket.accept()
86 nouvelle_connexion.sendall(bytes(str(N_bob), 'utf8'))
87 while 1:
88     message_chiffré_reçu = nouvelle_connexion.recv(1024)
89     if message_chiffré_reçu:
90         re_separateur = re.compile(r"[ ]+")
91         print("message chiffré reçu: ", message_chiffré_reçu)
92         message_chiffré_reçu_en_utf8 = message_chiffré_reçu.decode('utf8')
93         liste = re_separateur.split(str(message_chiffré_reçu_en_utf8))
94         message_chiffré_reçu_en_int = [int(x) for x in liste]
95         n_de_Alice = message_reçu_en_int[len(message_chiffré_reçu_en_int)-1] #récupérer n d'alice
96         message_déchiffré_en_int = [lpowmod(x, d_bob, N_bob) for x in message_chiffré_reçu_en_int[:len(message_chiffré_reçu_en_int)-1]]
97         message_déchiffré_hex = [hex(x) for x in message_déchiffré_en_int]
98         message_déchiffré_en_bytes = [bytes.fromhex(x[2:]) for x in message_déchiffré_hex]
99         message_déchiffré_utf8 = [x.decode('utf8') for x in message_déchiffré_en_bytes]
100        message_déchiffré = ''.join(message_déchiffré_utf8)
101        print("Voilà le message reçu déchiffré :", message_déchiffré)
102    if not message_chiffré_reçu:
103        break
104

```

Exécution :



```

hajar@hajar-ThinkPad-X240: ~
hajar@hajar-ThinkPad-X240: ~$ python3 Bob.py
En attente de message...
message chiffré reçu: b'19225919137641905542673476 24181091167565979312993941'
Voilà le message reçu déchiffré : bonjour

```

Quand Bob veut envoyer un message chiffré à Alice, il suit la même procédure qu'Alice lui-même a utilisée pour envoyer un message chiffré à Bob (**image- 1**). Alice, pour sa part, suit la même procédure que Bob a utilisée pour déchiffrer le message chiffré envoyé par Alice (**image- 2**). Ainsi, Alice et Bob peuvent s'envoyer des messages chiffrés avec RSA, qui seront déchiffrés par le destinataire. Ces messages sont envoyés de manière alternée.

```

110 while 1:
111     message_de_bob = input("message?")
112     if message_de_bob:
113         re_separateur = re.compile(r"[']+")
114         liste = re_separateur.split(str(message_de_bob))
115         liste_en_hex = [bytes(u,'utf8').hex() for u in liste]
116         conv_to_int = [int(x,16) for x in liste_en_hex]
117         message_chiffré_par_mot = [str(powmod(x,e,n_de_Alice)) for x in conv_to_int]
118         message_chiffré = ''.join(message_chiffré_par_mot)
119         nouvelle_connexion.sendall(bytes(message_chiffré,'utf8'))
120         print("Voilà le message chiffré envoyé:",message_chiffré)
121     if not message_chiffré_reçu:
122         break
123     ma_socket.close()
124
125

```

Images-1

```

108 while 1:
109     message_chiffré_reçu = ma_socket.recv(1024)
110     if message_chiffré_reçu:
111         re_separateur = re.compile(r"[']+")
112         print("message chiffré reçu: ",message_chiffré_reçu)
113         message_chiffré_reçu_en_utf8 = message_chiffré_reçu.decode('utf8')
114         liste = re_separateur.split(str(message_chiffré_reçu_en_utf8))
115         message_chiffré_reçu_en_int = [int(x) for x in liste]
116         message_déchiffré_en_int = [powmod(x,d_alice,N_alice) for x in message_chiffré_reçu_en_int]
117         message_déchiffré_hex = [ hex(x) for x in message_déchiffré_en_int]
118         message_déchiffré_en_bytes = [bytes.fromhex(x[2:]) for x in message_déchiffré_hex ]
119         message_déchiffré_utf8 = [x.decode('utf8') for x in message_déchiffré_en_bytes]
120         message_déchiffré = ''.join(message_déchiffré_utf8)
121         print("Voilà le message reçu déchiffré :",message_déchiffré)
122     if not message_chiffré_reçu:
123         break
124     ma_socket.close()

```

Image - 2

## 5-les problèmes rencontrés :

```

-----
N_bob = int(ma_socket.recv(1024))
while 1:
    message = input("message?")
    if message:
        re_separateur = re.compile(r"[']+")
        liste = re_separateur.split(str(message))
        liste_en_hex = [bytes(u,'utf8').hex() for u in liste]
        conv_to_int = [int(x,16) for x in liste_en_hex]
        message_chiffré_par_mot = [str(powmod(x,e,N_bob)) for x in conv_to_int]
        message_chiffré = ''.join(message_chiffré_par_mot)
        message_chiffré_envoyé = message_chiffré + ' ' + str(N_alice)
        print("Voilà le message chiffré envoyé:",message_chiffré_envoyé)
        ma_socket.sendall(bytes(message_chiffré_envoyé, 'utf8'))
        if message == "fin":
            break # Sort de la boucle si Le message est "fin"
# Ajout de la condition pour terminer La boucle Lorsque aucun message n'est envoyé
    if not message:
        break

```

Boucle infinie dans le Programme d'Alice

Le problème rencontré est que, lors de l'exécution du programme de Bob, celui-ci reçoit un message chiffré d'Alice, le déchiffre et l'affiche, mais ne demande pas un nouveau message à l'utilisateur via l'instruction « **message\_de\_bob = input ("message ?")** ». Ainsi, Bob ne peut pas envoyer de nouveau message à Alice.

Le problème semble être lié au fait que la boucle infinie dans le programme d'Alice ne se termine jamais. La boucle ne vérifie pas si le message reçu est égal à "fin" et ne sort donc jamais de la boucle, même si ce message est envoyé par Bob. On a essayé plusieurs méthodes mais on n'a pas réussi à résoudre ce problème.

## 7-Conclusion

Ce projet nous a permis de mieux comprendre le fonctionnement du protocole TCP avec les sockets et d'apprendre plus de choses sur python. Nous avons appris à utiliser les sockets pour créer des connexions entre deux programmes et à utiliser le protocole TCP pour envoyer et recevoir des données de manière fiable. Nous avons également appris à utiliser différentes fonctions de python pour manipuler les données et à travailler avec des expressions régulières. En outre, nous avons également appris à utiliser la cryptographie RSA pour chiffrer et déchiffrer les messages échangés entre les programmes. En résumé, ce projet nous a permis de développer nos compétences en python et en réseau informatique.

## 8-Bibliographie

[https://p-fb.net/master1/res\\_sys/cours/cours\\_Res\\_Sys\\_reseau.pdf](https://p-fb.net/master1/res_sys/cours/cours_Res_Sys_reseau.pdf)  
<https://python.jpvweb.com/mesrecettespython/doku.php?id>