

**Ecole supérieure en science et technologie de l'informatique
et du numérique ESTIN**

Compte rendu TP 04 _1

Chiffrement asymétrique

Réalisé par :

Benmedakhene Souha (G2) .

Encadré par :

Dr . Bouchoucha Lydia

Année universitaire : 2023/2024

Introduction :

L'algorithme **RSA**, développé par Ron Rivest, Adi Shamir et Leonard Adleman, est une technique de cryptographie asymétrique largement utilisée. Il repose sur l'utilisation de paires de clés, une **publique** et une **privée**, pour assurer la confidentialité et l'authenticité des données échangées sur des réseaux non sécurisés. RSA exploite des concepts mathématiques complexes comme la factorisation d'entiers pour garantir la sécurité des communications. RSA demeure un outil essentiel pour la protection des données dans un monde numérique en constante évolution, utilisé dans divers domaines tels que les transactions bancaires en ligne, la sécurisation des e-mails et la protection des informations sensibles gouvernementales et commerciales.

Objectifs :

Les objectifs de ce travail pratique sont les suivants :

1. Comprendre les principes fondamentaux de l'algorithme de **chiffrement** RSA, y compris le concept de clés publiques et privées, ainsi que les opérations de chiffrement et de déchiffrement.
2. Acquérir une expérience pratique dans la génération de paires de clés RSA et dans l'utilisation de ces clés pour chiffrer et déchiffrer des messages.
3. Expliquer le processus de chiffrement et de déchiffrement RSA .
4. Mettre en œuvre les étapes de génération de clés, de chiffrement et de déchiffrement RSA en utilisant un langage de programmation tel que **Python**.
5. Valider l'exactitude de l'implémentation en testant avec différents messages et en vérifiant que les résultats correspondent aux attentes théoriques .

Partie 1 :

L'algorithme RSA est un pilier de la cryptographie moderne, offrant une méthode robuste pour sécuriser les communications sur des réseaux non sécurisés. Dans cet environnement, Alice et Bob utilisent des paires de clés RSA pour échanger des messages de manière sécurisée. Dans cette section, nous examinerons comment Alice et Bob utilisent l'algorithme RSA pour garantir la confidentialité et l'authenticité de leurs communications.

a- quel est le service de sécurité assuré par le schéma décrit ci-dessous ?

réponse :

Le service de sécurité assuré par le schéma décrit est la confidentialité, l'intégrité, l'authenticité et la non-répudiation des communications et du stockage de données électroniques.

b- quelle clé Bob doit-il utiliser pour envoyer un message MSG chiffré à Alice ?

réponse :

Pour envoyer un message chiffré à Alice, Bob doit utiliser la clé publique d'Alice, $\{n_A, e_A\}$.

c- quelle clé Alice doit- il utiliser pour décrypter un message provenant de Bob?

réponse :

Pour décrypter un message provenant de Bob, Alice doit utiliser sa clé privée, $\{d_A, n_A\}$.

d- Alice peut-il se rendre compte que le message MSG chiffré provient de Bob ?

réponse :

Alice peut vérifier si le message chiffré provient de Bob en utilisant la clé publique de Bob pour déchiffrer le message. Si le déchiffrement réussit, cela signifie que le message a été chiffré avec la clé privée de Bob. ou bien en utilisant la signature numérique .

e- quelles clés Alice doit-il utiliser pour signer et chiffrer un document pour Bob ?

réponse :

Pour signer et chiffrer un document pour Bob, Alice doit utiliser sa propre clé privée $\{d_A, n_A\}$ pour signer le document et la clé publique de Bob $\{n_B, e_B\}$ pour chiffrer le document.

f- quelles clés Bob doit- il utiliser pour déchiffrer le document et vérifier la signature ?

réponse :

Pour déchiffrer le document et vérifier la signature d'Alice, Bob doit utiliser sa clé privée $\{d_B, n_B\}$ pour déchiffrer le document et la clé publique d'Alice $\{n_A, e_A\}$ pour vérifier la signature.

Partie 2 :

Maintenant que nous avons exploré les aspects pratiques de l'utilisation de l'algorithme RSA par Alice et Bob pour sécuriser leurs communications, nous allons plonger dans l'essence même de l'algorithme RSA en écrivant son implémentation. L'algorithme RSA repose sur des principes mathématiques complexes pour garantir la sécurité des communications. En rédigeant le pseudo-code de l'algorithme de chiffrement RSA, nous allons décomposer le processus en étapes élémentaires, mettant en évidence les calculs impliqués. Cette compréhension approfondie nous permettra de mieux implémenter l'algorithme dans un langage de programmation tel que Python et de l'appliquer dans des scénarios réels de sécurisation des communications .

Implémentation :

Etape 1: Génération des clés

La génération de clés dans RSA est une étape cruciale dans la mise en place d'un système de cryptographie sécurisé. Elle implique la création de deux clés distinctes mais liées mathématiquement : une clé publique et une clé privée. Ces clés sont générées à partir de

grands nombres premiers et suivent un processus complexe garantissant leur robustesse et leur sécurité.

. Voici une explication plus détaillée de cette étape :

1. Choix de nombres premiers :

- Deux nombres premiers distincts et de grande taille, généralement de taille similaire, sont choisis aléatoirement. Ces nombres sont conventionnellement notés p et q .

2. Calcul de n et ϕ :

- Le produit n de p et q est calculé : $n = p \times q$.
- La fonction d'Euler ϕ est calculée : $\phi = (p-1) \times (q-1)$.

3. Choix de e :

- Un entier e est choisi aléatoirement tel que $1 < e < \phi$, et que e et ϕ n'ont pas de diviseurs communs autres que 1 (c'est-à-dire $\text{pgcd}(e, \phi) = 1$).

4. Calcul de d :

- L'algorithme d'Euclide étendu est utilisé pour calculer l'entier d tel que $d \times e \equiv 1 \pmod{\phi}$, c'est-à-dire $(d \times e) \bmod \phi = 1$. d est l'inverse modulaire de e modulo ϕ .

5. Formation des clés :

- La clé publique est formée par le couple (n, e) .
- La clé privée est l'entier d .

Et voici le code en Python :

- on a utilisé la bibliothèque **random** pour générer des nombres aléatoires, **math** pour utiliser des opérations mathématiques comme mod , div , pgcd et **sympy** pour utiliser l'algorithme extended euclidean.
- **randrange** est une fonction en Python utilisée pour générer un entier aléatoire dans une plage donnée.
- **gcd** est une fonction en Python utilisée pour calculer le pgcd de 2 nombres.

```
RSA_algo.py 1
RSA_algo.py > generate_keypair
1
2 import random
3 import math
4 from sympy import randprime, mod_inverse
5
6 # choisir des nombres aleatoires
7
8 p = randprime(1, 100)
9 q = randprime(1, 100)
10
11 def generate_keypair(p, q):
12     n = p * q
13     phi = (p-1) * (q-1)
14
15     # Choisir une valeur aleatoire de e inferieur a phi
16     e = random.randrange(1, phi)
17
18 # verifier que le pgcd de e et phi egale a 1
19 g = math.gcd(e, phi)
20 while g != 1:
21     e = random.randrange(1, phi)
22     g = math.gcd(e, phi)
23
24 # gener le nombre d en utilisant the extended euclidean algorithm
25
26 d = mod_inverse(e, phi)
27
28
29
```

```
RSA_algo.py 1
RSA_algo.py > ...
11 def generate_keypair(p, q):
29
30     # verifier que d est entre 1 et phi
31     assert 1 < d < phi
32     # verifier que le produit de (e , d ) mod phi egale a 1
33
34     assert (d * e) % phi == 1
35
36     return ((e, n), (d, n))
37
38 # affichage des valeurs
39
40 print(" p = " ,p)
41
42
43 print(" q = " , q)
44
45 public_key, private_key = generate_keypair(p, q)
46
47 print(" The public key is : " , public_key , "The private key is : " , private_key)
48
49
50
```

Etape 2: Chiffrement

L'étape de chiffrement dans l'algorithme RSA est cruciale dans le processus, car elle permet de sécuriser les communications en convertissant les données en clair en données chiffrées à l'aide des clés de chiffrement générées précédemment (la clé publique et la clé privée). Cette étape est essentielle pour garantir la confidentialité des données échangées. Voici une explication détaillée de chaque aspect de l'étape de chiffrement :

1. **Génération des clés RSA:** Le destinataire génère une paire de clés - une clé publique et une clé privée.

2. **Représentation du message:** L'expéditeur représente le message sous forme d'entier dans une plage spécifique.
3. **Chiffrement RSA:** L'expéditeur chiffre le message à l'aide de la clé publique du destinataire en utilisant la formule $c = m^e \bmod n$.
4. **Transmission sécurisée:** Le message chiffré est transmis au destinataire sur un canal de communication non sécurisé, en garantissant que seuls le destinataire, avec la clé privée correspondante, peut déchiffrer et accéder au message d'origine.

Et voici le code en Python :

- La fonction **Pow** en Python est utilisée pour élever un nombre à une puissance. Elle prend deux arguments : le nombre de base et l'exposant.
- et le '%' pour calculer le **Mod**.

```
def encrypt(message, public_key):
    n, e = public_key
    # Ensure message is within the interval [0, n-1]
    if message < 0 or message >= n:
        raise ValueError("Message is not within the interval [0, n-1]")
    # Compute ciphertext
    ciphertext = pow(message, e) % n
    return ciphertext

# Encrypt the message for A
ciphertext = encrypt(message, public_key)
# Send the ciphertext to A
print("Ciphertext:", ciphertext)
```

Etape 3: Déchiffrement

1. **Récupération des clés:** Le destinataire utilise sa clé privée, composée de deux éléments : l'exposant de déchiffrement d et le module n .
2. **Déchiffrement du message chiffré:** Le destinataire reçoit le message chiffré c . Pour déchiffrer le message, le destinataire utilise sa clé privée en effectuant le calcul suivant : $m = c^d \bmod n$ où m est le message original.
3. **Représentation du message déchiffré:** Une fois le calcul effectué, le destinataire obtient le message original m sous forme d'entier dans la plage spécifiée.

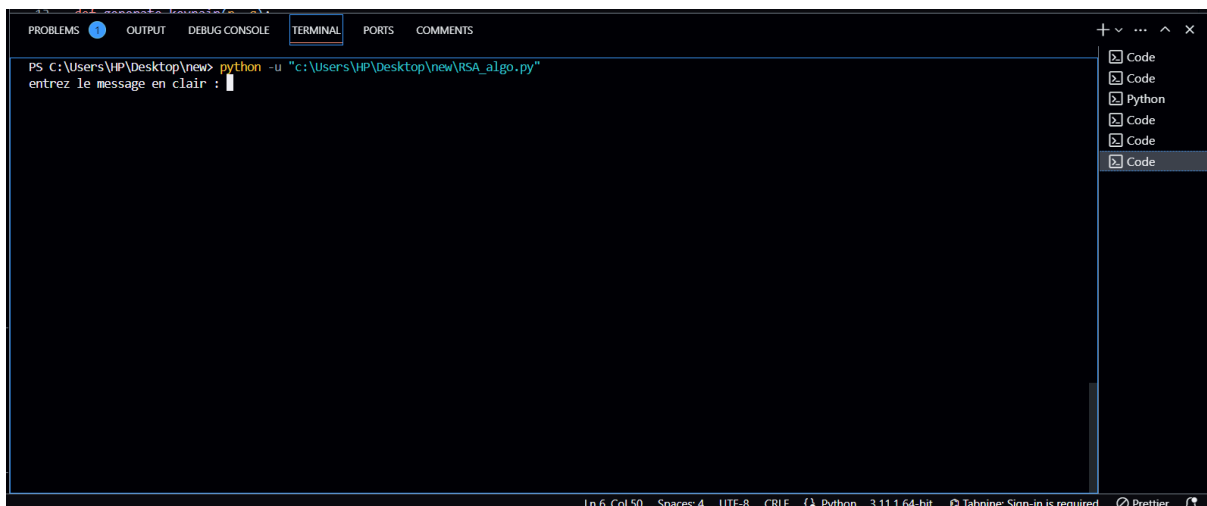
Et voici le code en Python :

```
74
75 def decrypt(ciphertext, private_key):
76     d, n = private_key
77     plaintext = pow(ciphertext, d) % n
78     return plaintext
79
80 private_key = private_key
81 recovered_message = decrypt(ciphertext, private_key)
82 print("Recovered message:", recovered_message)
```

Etape 4: Test et Validation

La validation est une étape essentielle dans le développement d'un algorithme de chiffrement tel que RSA. Elle garantit que le processus de chiffrement fonctionne comme prévu et offre un niveau de sécurité suffisant. Pendant les tests, différents exemples de texte en clair sont chiffrés en utilisant différentes clés publiques et privées. Cela permet de confirmer que l'algorithme de chiffrement RSA fonctionne de manière précise dans différentes situations et offre la sécurité nécessaire pour protéger les communications sensibles.

Dans le cadre de cette validation, je vais tester l'algorithme en utilisant le message en clair fourni par l'utilisateur dans cet exemple le message = 13 .



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal displays the command `python -u "c:\Users\HP\Desktop\new\RSA_algo.py"` and the prompt `entrez le message en clair :` with a cursor. The terminal is part of a workspace with several files open, including `Code`, `Code`, `Python`, `Code`, `Code`, and `Code`. The status bar at the bottom indicates the current file is at line 6, column 50, with 4 spaces, UTF-8 encoding, and CRLF line endings. The Python version is 3.11.1 64-bit, and there is a note about Tabnine sign-in.


```
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\RSA_algo.py"
entrez le message en clair : 13
```

et voici l'affichage final où nous obtenons **message = recovered message** , démontrant ainsi le bon fonctionnement de l'algorithme.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\RSA_algo.py"
entrez le message en clair : 13
p = 11
q = 79
The public key is : (629, 869) The private key is : (749, 869)
Ciphertext: 183
Recovered message: 13
PS C:\Users\HP\Desktop\new>
```

Conclusion :

En conclusion, cette expérience pratique nous a immergés dans le domaine de la cryptographie asymétrique en explorant en profondeur l'algorithme RSA. Nous avons acquis une compréhension approfondie des principes fondamentaux de RSA, incluant la génération de paires de clés, le processus de chiffrement et de déchiffrement, ainsi que les meilleures pratiques de sécurité associées. En implémentant l'algorithme RSA avec Python,

nous avons pu observer concrètement son fonctionnement et sa capacité à sécuriser les communications sur des réseaux non sécurisés.