TP 8 - Cryptographie appliquée

TP Réseaux Avancés M1 SIC - IA

Ilyas Bambrik

2024

Table des matières

Introduction 3

I - **Exercice : Fonction de hashage** 4

II - **Exercice : Chiffrement avec DES et AES (Algorithmes de chiffrement symétriques)** 7

III - **Exercice : Génération de clé RSA et chiffrement/déchiffrement** 11

IV - **Exercice : Communication avec serveur sécurisée** 13

V - **Encodage Base 64** 16

VI - **Exercice : Bibliothèque base64** 19

VII - **Exercice : Communication crypté AES avec encodage base64** 20

VIII - **Exercice : Chiffrement et Déchiffrement AES et base64** 23

IX - **Exercice : Communication Client-Serveur** 25

Introduction

La cryptographie est un sous domaine de la *cryptologie* qui consiste à appliquer des opérations mathématiques permettant de cacher un texte claire des parties non autorisées. Actuellement, cette technologie est utilisée dans tout domaine où le contenu de l'information doit être protégé (téléphonie, carte de crédit, applications comme les navigateurs web,etc).

Sous domaines de la cryptographie :

- *Symetric Cryptography :* La même clé est utilisée pour chiffrer et déchiffrer (*comme AES, DES et TripleDES*). Dit aussi *private key encription*.
- *Asymetric Cryptography :* La clé utilisée pour le chiffrement est différente de celle utilisée pour le déchiffrement (*comme RSA et Elliptical curve*). Dit aussi *public key encryption*. La clé privée est gardée secrète pour le déchiffrement alors que la clé publique est accessible par n'importe qui. Cependant, a partir de la clé publique, il est impossible de construire la clé privé.

Exercice : Fonction de hashage

I

- Nous avons vus à plusieurs reprise un champ checksum (entête IP et TCP) permettant de vérifier si le contenu du paquet n'a pas été corrompu.
- La fonction permettant de calculer le cheksum (aussi connue sous le nom CRC) est une fonction de hashage. Le hashage produit est considéré comme une emprunte qui permet de vérifier l'intégrité du contenu par le récepteur.
- Plusieurs algorithmes de hashage existent actuellement : MD5, SHA, CRC. La bibliothèque standard python inclue ces fonctions dans la librairie hashlib. Ces algorithmes ressemblent beaucoup aux algorithmes cryptographie AES et DES.
- Chaque algorithme produit une valeur de hashage de taille fixe quelque soit la taille des données en entrée. Par exemple SHA256 produit une valeur de taille 256 bits (32 octets), SHA1 produit des valeurs de 160 bits (20 octets) et MD5 avec 128 bits.

Remarque : Dans un site web dynamique, les mots de passe des utilisateurs sont souvent enregistrés sous forme de hash au lieu de leurs valeurs claires.

```
1 # importe les fonctions sha256, sha1 et crc32
2
3 from hashlib import sha256
4 from hashlib import sha1
5 from zlib import crc32
6
7 # crée un message encode en octets (le b au debut
8 # de la chaine de caracteres signifie byte)
9
10 message=b"Un message a transporter"
11
12 # calcule et affiche le hashage du message
13 # avec chaque algorithme
14
15 print("sha256 =",sha256(message).hexdigest().upper())
16 print("sha1 =",sha1(message).hexdigest().upper())
17 print("crc32 =",hex(crc32(message))[2::].upper())
```

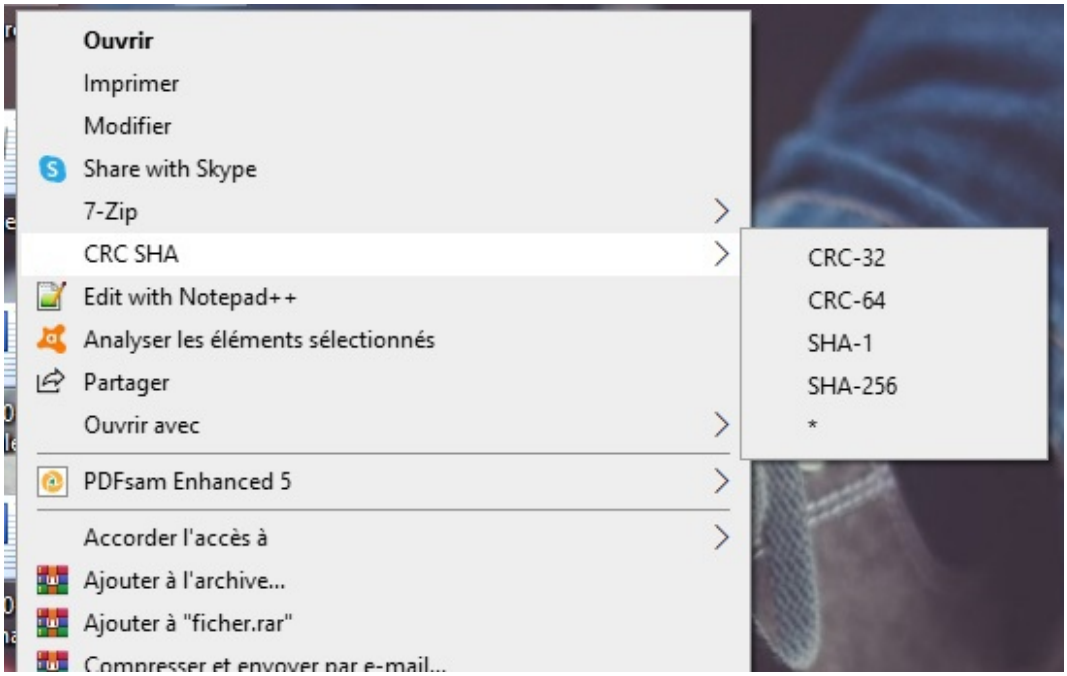
La sortie du hashage de "Un message a transporter" est (l'affichage est en hexadécimale):

```
sha256 = 16F71232D915F3533D6510E032EB1C4BFEF2AF39E3EB44A0B58B9C44D4A65361
```

```
sha1 = 2B3A855FBA7F4A19090DCB61AC5C7011E7D12AFC
```

```
crc32 = CB554B3E
```

Afin de vérifier bien que le résultat est juste, Windows offre un outil pour calculer le hashage d'un fichier (cliquez sur le bouton droit sur le fichier pour le quel vous souhaitez calculer le hash). Voir la figure suivante :



Sinon, des outilles online existent qui permettent de calculer le hashage d'un fichier entier ou d'un message texte(voir *https://www.fileformat.info/tool/hash.htm*).

Important : Si vous décider de comparer les valeurs produit par python et ceux de Windows ou FileFormat, assurez vous que le message soit enregistrer dans un fichier texte claire et non .doc ou .pdf .

Question

1. Écrivez un programme permettant de lire un fichier comme une suite d'octets (de préférence le fichier ne doit pas dépasser 1Mo).
2. Concaténez les octets lus dans une variable appelée *data*.
3. Calculez le hash SHA256 et MD5 de la variable *data* et comparez le résultat avec celui produit avec le site Fileformat. (enregistrez le résultat du hash afin de comparer avec l'étape suivante)
4. Changez un seul caractère du Contenu claire et comparez le résultat produit après modification avec le résultat du hashage avant modification. *Que remarquez vous ?*
5. *Est ce que deux messages / fichiers différent peuvent avoir le meme résultat de hashage ?*

Indice :

- Utilisez open("votre nom de fichier","rb"). "rb" signifie : r=> ouvrir en mode lecture , b=> lire des suites d'octets au lieu de texte ASCII
- Pour initialiser une suite d'octets *data* vides (type *bytes*), il suffit de mettre :
- data=b''
- Pour concaténer une séquence avec une autre, il suffit de les sommer comme une chaîne de caractères en Java.

Le programme suivant propose un exemple sur comment lire le contenu du fichier *res.docx* sous forme d'octets :

```
1 fichier=open("res.docx","rb")
2 data=b''
3 for sequence_octets in fichier:
4         data+=sequence_octets
5 fichier.close()
```

Le code suivant permet de lire le fichier entier avec une seule instruction :

```
1 data=open("res.docx","rb").read()
```

II

Exercice : Chiffrement avec DES et AES (Algorithmes de chiffrement symétriques)

La bibliothèque Crypto inclue dans Anaconda offre l'implémentation de plusieurs algorithmes de chiffrement comme, DES, AES et RSA. Pour commencer, il faudra générer une clé primaire aléatoire selon la taille de la clé supportée par l'algorithme. Ce-ci peut être assuré par la fonction suivante

```
1 import os
2 # os.urandom(N) genere une sequence
3 # de N octets aleatoire
4
5 # cle de 16 octets (128 bits) aletoire
6 cle_16_octet=os.urandom(16)
```

Le code suivant montre comment chiffrer un contenu avec DES en utilisant la bibliothèque de cryptographie *Crypto :*

```
1 from Crypto.Cipher import DES
2 from Crypto import Random
3 # affectation de la cle 8 octets
4 key = b'12345678'
5 # creation d'un vecteur d'initialisation (iv) aleatoire de 8 octets
6 iv = Random.new().read(DES.block_size)
7 # creation de l'objet de chiffrement DES (cipher)
8 cipher = DES.new(key, DES.MODE_CBC, iv)
9 # text claire
10 plaintext = b'sona si latine loqueris '
11 # message chiffre concatene avec le vecteur d'initialisation
12 # cipher.encrypt(plaintext) permet de chiffrer le contenu "plaintext"
13 # avec l'objet "cipher"
14 msg = iv +cipher.encrypt(plaintext)
15 #affichage du text chiffre
16 print("Message chiffré ="+str(msg))
17 #affichage du text dechiffre
18 print("Message déchiffré ="+str(cipher.decrypt(msg)))
```

Le code suivant montre comment chiffrer un contenu avec AES en utilisant la bibliothèque de cryptographie *Crypto :*

```
1 #importe AES
2 from Crypto.Cipher import AES
3 #cree une instance AES avec une cle= "cle16 octets AES"
4 #la taille de la cle ==  (16octets)
```

```
5 objet_de_chiffrement=AES.new("cle16 octets AES")
6 Message_claire="Message claire16"
7 #chiffre le message claire
8 contenu_chiffre=objet_de_chiffrement.encrypt(Message_claire)
9 print(contenu_chiffre)
```

Résultat :

```
b'|Kk\x7f\\Y\xb8\x1b\xb2J\xe3\xf0\xe3\xd0[8'
```

Important:

- Le texte à chiffrer doit être d'une taille multiple de la taille de la clé. Sinon des octets de bourrage doivent être ajoutés à la fin du contenu.
- Pour indiquer le nombre des octets de bourrage à la fin du contenu, il existe plusieurs méthodes :

    1. Ajout de la taille du message original sans bourrage avant le contenu chiffré.
    2. Ajout du nombre d'octets de bourrage avant le contenu chiffré.
    3. Utiliser un caractère absent du contenu claire pour les octets de bourrage afin de marquer la fin lors du déchiffrement.

- Lors de l'initialisation de l'algorithme (ligne 5), AES automatiquement obtient la taille de la clé (128 bits 192 bits ou 256 bits) pour déterminer la taille du bloque.

Le déchiffrement se fait d'une manière similaire. Une instance de l'algorithme AES avec la même clé est créée et la méthode *decrypt* est invoquée sur le contenu chiffré :

```
1 objet_de_dechiffrement=AES.new("cle16 octets AES")
2 #contenu_chiffre est le contenu produit lors du chiffrement
3 #dans les instruction du programme précedant
4 text_claire=objet_de_dechiffrement.decrypt(contenu_chiffre)
5 print(text_claire)
```

Résultat :

```
b'Message claire16'
```

Question 1

Écrivez un programme qui chiffre le contenu d'un fichier avec AES 128bits (16 octets) et l'enregistre :

1. Écrivez un programme permettant de lire le contenu d'un fichier sous forme de séquence d'octets. (voir le premier exercice)
2. Le programme doit calculer la taille en nombre d'octets du contenu.
3. A partir de la taille, le programme doit calculer le nombre d'octets de bourrage à ajouter à la fin (entre 0 et 15). Cette valeur doit être sauvegardée dans une variable *bourrage*.
4. Ajoutez les octets de bourrage selon la valeur obtenue.
5. Générez une clé de 16 octets avec *os.urandom*. Sauvegardez la clé dans un fichier *key.bin* afin de l'utiliser lors de la question de déchiffrement.
6. Chiffrez le contenu avec AES.
7. Ajoutez un octet au début du contenu chiffré contenant la valeur de *bourrage.*
8. Enregistrez le résultat dans un fichier.

Indice :

- Utilisez la fonction *len* pour obtenir la taille de la séquence d'octets. Afin d'obtenir le nombre d'octets de bourrage il suffit d'utiliser *l'opérateur modulo (%) avec 16*.
- Pour ouvrir le fichier avec le nom *key.bin* en mode écriture, il suffit d'utiliser *open("key.bin","wb")*. N'oubliez pas de fermer le fichier avec la méthode *close()* à la fin l'exécution.

Question 2

Implémentez l'algorithme permettant de déchiffrer le contenu du fichier que vous venez de chiffrer.

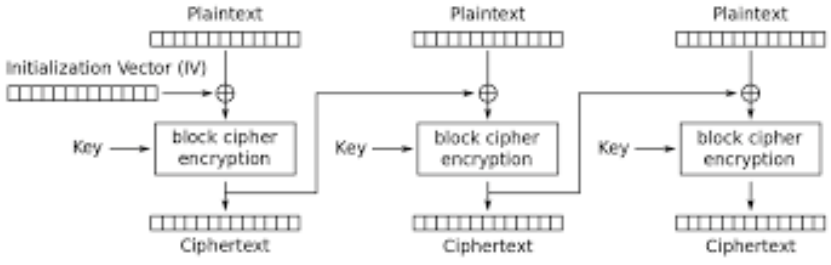
Vecteur d'Initialisation (IV) et Cipher-Block Chaining (CBC)

Avec l'implémentation précédente AES, l'algorithme chiffre chaque bloque séparément. Ainsi, une attaque possible est de permuter les positions des bloques de 128 bits.

Une solution à ce problème est le schéma Cipher-Block Chaining (CBC). Dans ce mode d'opération le premier bloque du contenu est d'abord combiné avec un Vecteur d'Initialisation (IV) par un XOR avant que le résultat soit passé en entrée des opérations de AES (SubByte). Par la suite, le résultat du chiffrement est utilisé comme IV pour le bloque suivant (le prochain bloque XOR le bloque chiffré précédant sera passé à SubByte).

Remarque : Le vecteur IV doit être de la même taille du bloque.

Ce fonctionnement est illustré dans la figure suivante :



Cipher Block Chaining (CBC) mode encryption

L'implémentation AES offre une façon simple pour exécuter ce traitement. Il suffit de passer en paramètre la valeur de IV initiale et le mode CBC :

```
#importe AES
from Crypto.Cipher import AES
#cree une instance AES avec une cle= "cle16 octets AES"
#l'argument AES.MODE_CBC indique que le chiffrement est
# en mode CBC

#le vecteur d'initialisation = "vecteur d'init16"

objet_de_chiffrement=AES.new("cle16 octets AES",AES.MODE_CBC,"vecteur d'init16")
Message_claire="Message claire16"
#chiffre le message claire
contenu_chiffre=objet_de_chiffrement.encrypt(Message_claire)
print(contenu_chiffre)

#dechiffrement avec CBC
objet_de_dechiffrement=AES.new("cle16 octets AES",AES.MODE_CBC,"vecteur d'init16"
)

text_claire=objet_de_dechiffrement.decrypt(contenu_chiffre)
print(text_claire)

```

Remarque : Il est préférable que IV soit aussi aléatoire comme la clé.

Question 3

Dans les étapes suivantes, on souhaite comparer entre le chiffrement AES 128 bits avec et sans CBC :

1. Créez une chaîne d'octets contenant au moins 32 octets (si la taille est supérieure à 32, elle doit être multiple de 16). Vous pouvez utiliser *os.urandom* ou une chaîne de texte. (pour convertir une chaîne de caractère en octets, utilisez la méthode *bytes* ou *encode()*)
2. Chiffrez le message avec AES 128bits sans CBC (comme dans le premier exemple AES de cette série).
3. Générez un vecteur d'initialisation de 16 octets contenant seulement des 0 (il suffit de mettre *bytes(16)*). Mettez cette valeur dans la variable iv.
4. Chiffrez le même message avec AES 128bits et CBC (comme dans l'exemple AES avec CBC de cette série).
5. Comparez les deux résultats de chiffrement. Est ce que le premier bloque chiffré de 16 octets est identique pour les deux chiffrements ? Est ce que le deuxième bloque est le même pour les deux chiffrements ?

Exercice : Génération de clé RSA et chiffrement /déchiffrement

III

- Contrairement à AES, RSA est un algorithme de chiffrement asymétrique.
- Celui-ci nécessite la génération d'une clé publique *(ligne 19)* et d'une clé privée *(ligne 8)*.
- La clé publique est sauvegardée (*ligne 23*) dans `fichier_cle_publique.pem`. La clé privée est sauvegardée (*ligne 11*) dans `fichier_cle_prive.pem` .
- Le programme suivant illustre comment générer des clés RSA.

```
1 # importe le module RSA
2 from Crypto.PublicKey import RSA
3
4 # genere une cle publique/prive avec modulos (N) de 2048 bits
5 cle= RSA.generate(2048)
6
7 # recupere la cle prive
8 cle_prive = cle.exportKey()
9
10 #ouvre un fichier en mode d'ecriture en octets
11 Fichier = open("fichier_cle_prive.pem", "wb")
12
13 # ecrit le contenu de la cle prive sur fichier
14 Fichier.write(cle_prive)
15 # ferme le fichier
16 Fichier.close()
17
18 # recupere la cle publique
19
20 cle_publique = cle.publickey().exportKey()
21
22
23 # ecrit le contenu de la cle publique sur fichier
24 Fichier = open("fichier_cle_publique.pem", "wb")
25 Fichier.write(cle_publique)
26 Fichier.close()
```

Listing 1 `Création d'une clé publique/privée`

- Le programme suivant permet d'importer les clés publique *(ligne 5)* et privée *(ligne 20)* ainsi que le chiffrement avec clé publique *(ligne 14)* et déchiffrement avec clé privée *(ligne 26)*.
- Un objet permettant de chiffrer avec la clé publique (*ligne 8*) est affecté à *objet_cle_rsa*.
- Un objet permettant de chiffrer avec la clé privée (*ligne 23*) est affecté à *objet_cle_rsa_prive*.

```
1 import Crypto
```

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import  PKCS1_OAEP
# importe la cle publique du fichier
contenu_clepublique = RSA.importKey(open("fichier_cle_publique.pem","rb").read())

# cree un objet a partir de la cle permetant de chiffre avec RSA
objet_cle_rsa = PKCS1_OAEP.new(contenu_clepublique)

# message claire
message=b"Message"

# chiffre le message avec la cle publique
message_chiffre=objet_cle_rsa.encrypt(message)

#imprime le resultat du chiffrement
print(message_chiffre)

# importe la cle prive du fichier
contenu_cleprive = RSA.importKey(open("fichier_cle_prive.pem","rb").read())

# cree un objet a partir de la cle permetant de chiffre avec RSA
objet_cle_rsa_prive = PKCS1_OAEP.new(contenu_cleprive)

#imprime le resultat du dechiffrement
print(objet_cle_rsa_prive.decrypt(message_chiffre))
```

Listing 2 `Chiffrement avec clé publique et déchiffrement avec la clé privée`

Question

Pour un exercice échauffement, créez un programme permettant de déchiffrer avec RSA le fichier *1_crypte.txt* situé dans le lien suivant. La clé privée permettant le déchiffrement du contenu est situé dans le même répertoire Google *fichier_cle_prive.pem*, utilisez celle-ci pour déchiffrer :

https://drive.google.com/drive/folders/1nR-WknZLy2PYOzlA6uptUCxw6ETFDaN3?usp=sharing

Exercice : Communication avec serveur sécurisée

IV

Pour commencer, vous devez générer les deux fichiers contenant la clé publique et la clé privée avec le programme expliqué dans *Listing 1*. La clé publique doit être placée dans le fichier *fichier_cle_publique.pem*, et la clé privé doit être placée dans le fichier *fichier_cle_prive.pem* .

Les deux programmes représentent un code du serveur muti-thread et du client. Ces derniers doivent être placés dans le même répertoire que les fichiers des clés publique et privée (*fichier_cle_publique.pem* et *fichier_cle_prive.pem*).

- Le serveur muti-thread (*Listing 3*) est le même que le serveur multi-thread du TP 1 mais transmet une clé publique au client initialement (*contenu_clepublique*) ;
- Par la suite, les messages reçus du clients doivent être déchiffrés avec la clé privée (*objet_cle_rsa_prive*)

```
import socket
import _thread as thread
from Crypto.PublicKey import RSA
from Crypto.Cipher import  PKCS1_OAEP


def Traiter_Connexion(connexion_avec_client,adresse_client):
   global objet_cle_rsa_prive,contenu_clepublique
   MessageRec=b""

   print ("Connexion de la machine = ", adresse_client)
   # Transmettre le contenu de la cle publique apres la connexion
   # Utilisez la methode send de connexion_avec_client pour
   # transmettre contenu_clepublique
   # (A FAIRE 1) ajoutez l'instruction dans cette ligne


   try:
      while True:
          MessageRec=connexion_avec_client.recv(1024)
          # Dechiffre le message recu (MessageRec)
          # Utilisez la methode decrypt de la cle prive (objet_cle_rsa_prive)
          # Affectez le resultat du dechiffrement a MessageRec
          # (A FAIRE 2) ajoutez l'instruction dans cette ligne


          if MessageRec==b"Fin":
                  break

```

```
31            print("Client" ,adresse_client," a dit :",MessageRec.decode())
32     except:
33        print("Deconnexion")
34     print("Deconnexion de :",adresse_client)
35     try:
36        connexion_avec_client.close()
37     except:
38        pass
39
40 SocketServeur = socket.socket()
41 host = socket.gethostname()
42 port = 9500
43 SocketServeur.bind(("127.0.0.1", port))
44
45 SocketServeur.listen(5)
46
47 contenu_clepublique = open("fichier_cle_publique.pem","rb").read()
48
49 contenu_cleprive = RSA.importKey(open("fichier_cle_prive.pem","rb").read())
50 # cree un objet a partir de la cle permetant de chiffre avec RSA
51 objet_cle_rsa_prive = PKCS1_OAEP.new(contenu_cleprive)
52 print("Lancement serveur")
53 while True:
54    ConnexionAUnClient, addrclient = SocketServeur.accept()
55    thread.start_new_thread(Traiter_Connexion,(ConnexionAUnClient,addrclient))
56
```

Listing 3. `Code Serveur multi-thread avec chiffrement`

- Après la connexion au serveur, le client (*Listing 4*) doit recevoir la clé publique qui sera transmise automatiquement par le serveur. A partir de la clé publique reçue (*Cle_publique*), le client crée un objet de clé publique (*objet_cle_rsa_publique*) ;
- Par la suite, pour chaque message saisie par le client, celui-ci est chiffré avec la clé publique (*objet_cle_rsa_publique*) et transmis (*resultat_chiffre*) ;

```
1 import socket
2 from Crypto.PublicKey import RSA
3 from Crypto.Cipher import  PKCS1_OAEP
4
5 SocketClient = socket.socket()
6 host = socket.gethostname()
7 port = 9500
8 SocketClient.connect(("127.0.0.1", port))
9
10 Cle_publique=b""
11 while True:
12         recu=SocketClient.recv(1024)
13         Cle_publique+=recu
14         if b'-----END PUBLIC KEY-----' in Cle_publique:
15                 break
16
17 # Importez la cle a partir du contenu recu Cle_publique
18 # Utilisez la methode importKey de RSA pour importer la cle
19 # Mettez le resultat dans clepublique
20 # Creez un objet objet_cle_rsa_publique a partir de clepublique (objet de cle
   publique)
21 # Utilisez PKCS1_OAEP.new pour faire ceci
22 # (A FAIRE 1) ajoutez deux instructions dans les deux lignes suivantes
```

```
23
24
25 print("Lancement serveur")
26 while True:
27    MessageATransmettre=input().encode()
28    # Chiffre le message lu du clavier (MessageATransmettre)
29    # Utilisez la methode encrypt de objet_cle_rsa_publique
30    # Mettez le resultat dans resultat_chiffre
31    # (A FAIRE 2) ajoutez l'instruction dans cette ligne
32
33
34
35    SocketClient.send(resultat_chiffre)
36    if(MessageATransmettre==b"Fin"):
37       break
38       print( "Deconnexion de :",addrclient)
39 ConnexionAUnClient.close()
```

Listing 4. `Code Client avec chiffrement`

Question

Complétez les instructions des programmes client/serveur là où il y a un commentaire # (A FAIRE). Vos objectifs sont les suivants :

1. *Programme Serveur :* Transmettre le contenu de la clé publique au client (A FAIRE1 - Ligne 16). *Une instruction suffirais pour ceci* ;
2. *Programme Client :* Créer l'objet de la clé publique au niveau du client [*clepublique*] à partir de la clé reçue du serveur [*Cle_publique*](A FAIRE 1-Ligne 21). Voir l'exemple dans Exercice 1 (Listing 2- Ligne 5) pour importer la clé à partir du contenu reçu ;
3. *Programme Client :* Créer un objet de la clé publique à partir de *clepublique*, et mettre le résultat dans l'objet *objet_cle_rsa_publique*. Voir Listing 2- Ligne 8 pour comment créer un objet à partir du contenu de la clé .(A FAIRE 1-Ligne 22) ;
4. *Programme Serveur :* Déchiffrer le contenu reçu du client [*MessageRec*] avec la clé privée [*objet_cle_rsa_prive*](A FAIRE 2-Ligne 25). *Une instruction suffirais pour ceci* ;
5. *Programme Client :* Chiffrer le contenu du message saisi (*MessageATransmettre*) par l'utilisateur avec la clé publique(*objet_cle_rsa_publique*). Mettez le résultat dans la variable *resultat_chiffre*. Voir Listing 2-Ligne 14 pour comment chiffrer. (A FAIRE 2-Ligne 32) ;

Remarque :

En réalité, le serveur transmet ça clé publique au client mais seulement afin que le client chiffre et transmet à son tour une clé symétrique (AES ou DES) au serveur. Par la suite, les messages transmis par le client et par le serveur sont chiffré avec la clé symétrique (et non pas par la clé publique RSA).

La difficulté de se programme réside dans comment encoder la langueur du message, calculer le nombre d'octets de bourrage ajoutés à la fin.

Encodage Base 64

V

Pour commencer, téléchargez les programmes de cette partie dans le lien suivant : *https://drive.google.com/drive/folders/1okQpQv641qJpvEgj8RRtPXZAjf_mUOAa?usp=sharing*

L'encodage Base 64 est un algorithme permettant de transformer des données binaires (image, sons, vidéo) en représentation ASCII (texte claire). Cet algorithme n'est pas un algorithme de chiffrement mais peut être utilisé pour encoder les données avant de les chiffrer.

- La première étape ajoute des octets 0 tel que le nombre d'octets dans le contenu à encoder est multiple de 3.
- Chaque bloque de 3 octets est divisé en bloques de 6bits.
- Chaque valeur de 6 bits est représentée par un caractère ASCII. Le bloque 6 bits peut prendre 2^6 (64 d'où le nom de l'encodage). Les 64 caractères sont les 10 chiffres, 26 minuscules, 26 majuscules , caractère + et /. Le 65em caractère = est utilisé pour marque les caractères de bourrage à la fin de l'encodage.
  Les correspondances entres les valeurs 6 bits et les caractères ASCII sont les suivantes :

Base64 Encoding Table							
Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Exemple : Exemple d'encodage Base 64

- Prenant le bloque de caractères suivant (4 octets):

ABCD

- La représentation binaire de ce bloque est la suivante :

01000001010000100100001101000100

- La taille du contenu n'est pas multiple de 3, ainsi 2 octets sont ajoutés (3 -(4%3)) à la fin. Le résultat est le suivant :

010000010100001001000011010001000000000000000000

- Diviser le contenu à encoder en bloques de 6 bits :

010000 010100 001001 000011 010001 000000 000000 000000

- Le résultat de l'encodage selon la table de correspondance est comme suite :

QUJDRA==

Remarque

Les séquences 6 bits avec valeur égale à 0 à la fin sont encodées avec =.

Les séquences 6 bits avec valeur égale à 0 au milieu du code sont encodées avec A.

Pour décoder un contenu représenté avec l'encodage 64, le processus inverse est appliqué :

1. Chaque caractère du code est représenté avec la valeur 6 bits correspondante (les espaces et les retours à la lignes sont automatiquement ignorés).
2. Les valeurs 6 bits sont concaténés dans un bloque de 24 bits.
3. Le bloque est ensuite divisé en octets de 8 bits pour récupérer le contenu originale.

Exercice : Bibliothèque base64

VI

La bibliothèque base64 de python permet d'encoder et de décoder du contenu base 64 :

```python
# importe la bibliotheque Base64
import base64
# message_a_encoder prends le contenu du message
# sous forme de sequence bytes

message_a_encoder="Votre contenu a encoder".encode()

# la methode b64encode de base64 permet d'encoder
# le parmetre du message est le message a encoder

message_encode=base64.b64encode(message_a_encoder)
print(message_encode)

# la methode b64decode de base64 permet d'decoder
# le parmetre du message est le message a decoder
message_decode=base64.b64decode(message_encode)
print(message_decode)
```

Question

1. Décodez la phrase suivante avec la méthode b64decode. Quel est le résultat ?
   QnJhdm8sIHZvdXMgYXZleiBkw6ljb2TDqSBsZSBtZXNzYWdlLg==
2. Ajoutez plusieurs caractères = à la fin du texte encodé précédant et puis essayer de décoder. Quel est l'effet sur le résultat ?

Exercice : Communication crypté AES avec encodage base64

VII

Le chiffrement avec AES est très simple avec la bibliothèque *Crypto*, mais lorsque le contenu chiffré n'est pas d'une taille multiple de 16 octets, la procédure ce complique. Le nombre d'octets de bourrage doit être ajouté au contenu afin que le récepteur les supprime.

Avec l'encodage Base 64, ce problème n'existe pas. Il suffit d'encoder le contenu à base 64 et d'ajouter autant de caractères = à la fin pour rendre la taille multiple de 16. Après, le contenu est encodé avec AES, et le récepteur peut repérer directement les octets de bourrage (les caractères = à la fin) sans avoir besoins d'autres informations.

```
#importe AES
from Crypto.Cipher import AES
import base64
#cree une instance AES avec une cle= "cle16 octets AES"
objet_de_chiffrement=AES.new("cle16 octets AES")

#ouvre le fichier et lit le contenu complet de celui-ci
# en mode bytes
Message_claire=open("fichier_claire.jpg","rb").read()

# (A Faire 1) utiliser base64 pour encoder le contenu
# du fichier Message_claire (base64.b64encode)
# Mettez le resultat dans la variable Message_Ecode


# si la taille du contenu n'est pas multiple de 16
if len(Message_Ecode)%16>0:
        # (A Faire 2) calculez le nombre d'octets de bourrage
        # pour que le contenu soit multiple de 16
        # et mettez cette valeur dans la variable padding


        # le contenu est complete avec le des octets b'='
        # repete avec padding fois
        Message_Ecode=Message_Ecode+padding*b'='
#chiffre le message encode
contenu_chiffre=objet_de_chiffrement.encrypt(Message_Ecode)

# creation d'un objet fichier en ecriture en mode bytes
fichier_sortie=open("crypte.jpg",'wb')
```

```
31 # ecriture du contenu chiffre
32 fichier_sortie.write(contenu_chiffre)
33 # fermeture du fichier
34 fichier_sortie.close()
35
```

Question 1

Complétez le programme précédant afin :

1. *A Faire 1 :* D'encoder le contenu avec Base64 (Ligne 11 à 14). Une instruction suffit.
2. *A Faire 2 :* De calculer le nombre d'octets de bourrage (b"=") qu'on doit ajouter à la fin pour que le chiffrement AES soit possible (Ligne 18 - 21). Une seule instruction suffit.

Le code de cette exercice est disponible dans le lien suivant :

https://drive.google.com/file/d/1eM3KbxHfsFxfxzjmB3H_qbxQd79KrFtV/view?usp=sharing

Le fichier "*fichier_claire.jpg*" est téléchargeable à partir du lien suivant (vous êtes aussi libre de chiffrer un fichier de votre choix, soyez sure que le fichier que vous essayez de chiffrer est dans le même répertoire que le programme) :

https://drive.google.com/file/d/1L7waoQVgFAZufQpBOUlVOVHJ8lO0t6vZ/view?usp=sharing

Question 2

Complétez le programme suivant afin de déchiffrer l'image suivante *(A FAIRE 1 et A FAIRE 2)* :

```
1 from Crypto.Cipher import AES
2 import base64
3 # cree une instance AES avec une cle= "cle16 octets AES"
4 objet_de_dechiffrement=AES.new("cle16 octets AES")
5 # lit le contenu chiffre a partir du fichier
6 contenu_chiffre=open("imagecrypte.jpg","rb").read()
7 # (A FAIRE 1) Dechiffre le contenu avec l'objet AES
8 # Utilisez la methode decrypt de objet_de_dechiffrement
9 # mettez le resultat dans la variable Message_Encode
10
11
12
13 # (A FAIRE 2) Decoder le contenu dechiffre avec base64
14 # Utilisez la methode b64decode
15 # Mettez le resultat dans la variable contenu_decode
16
17
18 # ouvre un fichier en ecriture
19 fichier_sortie=open("fichier_claire.jpg","wb")
20 fichier_sortie.write(contenu_decode)
21
22 fichier_sortie.close()
23
```

- Après avoir complété le programme (A FAIRE 1 et A FAIRE 2), quelle est l'image en résultat ?

Le code de cette exercice est disponible dans le lien suivant :

https://drive.google.com/file/d/1OYmp1Dg3dGvSs2bMaegwDb-POqNYvkE3/view?usp=sharing

Le fichier "*imagecrypte.jpg*" est téléchargeable à partir du lien suivant :

https://drive.google.com/file/d/1tixcjBQ1Q8Mi3aRhoiMplpJchn-LY7aA/view?usp=sharing

Exercice : Chiffrement et Déchiffrement AES et base64

VIII

Question 1

Complétez la procédure de chiffrement et hachage selon les instructions dans les commentaires A Faire 1,2,3 et 4

```
def procedureChifrement(message,key):
    # A FAIRE 1: calculez le hash md5 du message
    # placez le resultat dans la variable msgmd5 (utilisez la méthode hexdigest)

    print(msgmd5)
    # A FAIRE 2: encodez le message avec base64 et placez le dans msgenc

    print(msgenc)
    # A FAIRE 3:  calculer le padding
    # ajouter des b'=' à la fin de msgenc pour avoir une taille multiple de 8
    #print(len(msgenc),(8%len(msgenc)))

    print(msgenc)
    #print(len(msgenc),(8%len(msgenc)))
    # A FAIRE 4: chiffrez msgenc avec DES
    # utilisez la valeur key comme cle
    # placez le résultat dans msgencry

    print(msgencry)

    return msgmd5.encode()+b"\r\n"+msgencry
```

Question 2

Complétez la procédure de déchiffrement et de vérification de hachage selon les instructions dans les commentaires A Faire 1 et 2.

```
def procedureDechifrement(contenu,key):

    msgmd5,msgencry=contenu.split(b"\r\n")


    # A FAIRE 1:  dechifrez et decoder msgencry
    # placez le resultat dans messageclaire

    print(messageclaire)
```

```
10
11
12      # A FAIRE 2: verifier que msgmd5 est egale au hashage md5 de messageclaire
13      # si les deux valeurs sont differents retourner None
14
15
16      return messageclaire
```

Exercice : Communication Client-Serveur

IX

Le code Client et Serveur suivants utilisent RSA,AES et Base64 pour chiffrer les messages échangés.

```
import socket
from Crypto.PublicKey import RSA
from Crypto.Cipher import  PKCS1_OAEP
from Crypto.Cipher import AES
import base64
import time
import os
# cree une cle de 128 bits (16 octets)
cle_AES=os.urandom(16)


SocketClient = socket.socket()
port = 9500
SocketClient.connect(("127.0.0.1", port))

Cle_publique=b""
while True:
        recu=SocketClient.recv(1024)
        Cle_publique+=recu
        if b'-----END PUBLIC KEY-----' in Cle_publique:
                break

clepublique = RSA.importKey(Cle_publique)
# cree un objet a partir de la cle permetant de chiffre avec RSA
objet_cle_rsa_publique = PKCS1_OAEP.new(clepublique)


# chffre la cle AES avec la cle RSA

cle_AES_chiffre=objet_cle_rsa_publique.encrypt(cle_AES)

# (A FAIRE 1) envoyez la cle chiffre (cle_AES_chiffre) au serveur
# Utilisez la methode send de SocketClient


# met le programme en attente pour 5 secondes
time.sleep(5)

# cree l'objet de chiffrement AES
objet_de_chiffrement=AES.new(cle_AES)
```

```
41
42
43 while True:
44
45
46    MessageATransmettre=input().encode()
47    # (A FAIRE 2) Encodez la chaine de caracteres lue (MessageATransmettre)
48    # avec base64, mettez le resultat dans Message_Encode
49
50
51    # ajoute les octets b'=' a la fin pour que la taille soit
52    # multiple de 16
53    Message_Encode=Message_Encode+(16-(len(Message_Encode)%16))*b'='
54    # chiffre le contenu saisie par clavier avec la cle AES
55    SocketClient.send(objet_de_chiffrement.encrypt(Message_Encode))
56    if(MessageATransmettre==b"Fin"):
57       break
58       print( "Deconnexion de :",addrclient)
59 ConnexionAUnClient.close()
60
```

Listing 1. Code client

```
1 import socket
2 import _thread as thread
3 from Crypto.PublicKey import RSA
4 from Crypto.Cipher import  PKCS1_OAEP
5 from Crypto.Cipher import AES
6 import base64
7
8 def Traiter_Connexion(connexion_avec_client,adresse_client):
9    global objet_cle_rsa_prive,contenu_clepublique
10
11   print ("Connexion de la machine = ", adresse_client)
12   connexion_avec_client.send(contenu_clepublique)
13
14   # recoit la cle AES genere par le client chiffre
15   # avec la cle publique
16   cle_AES_chiffre=connexion_avec_client.recv(1024)
17   # (A FAIRE 1) Dechiffrez la cle AES (cle_AES_chiffre) avec la cle prive sur
   serveur
18   # Utilisez la methode decrypt de objet_cle_rsa_prive
19   # Mettez le resultat dans la variable cle_AES
20
21
22   # cree l'objet de dechiffrement AES
23   objet_de_dechiffrement=AES.new(cle_AES)
24   try:
25      while True:
26          Message_Chiffre=connexion_avec_client.recv(1024)
27          # (A FAIRE 2) Dechiffrez le message recu (Message_Chiffre) avec la cle
   AES
28          # Utilisez la methode decrypt de objet_de_dechiffrement
29          # Mettez le resultat dans la variable Message_Decode
30
31
32          # (A FAIRE 3) Decodez le resultat avec base64 (Message_Decode produit
   par l'instruction precedente)
33          # Utilisez la methode b64decode de base64
34          # Mettez le resultat dans la variable MessageRec
```

```
35
36
37          if MessageRec==b"Fin":
38                  break
39
40
41          print("Client" ,adresse_client," a dit :",MessageRec.decode())
42    except:
43       print("Deconnexion")
44    print("Deconnexion de :",adresse_client)
45    try:
46       connexion_avec_client.close()
47    except:
48       pass
49
50 SocketServeur = socket.socket()
51 host = socket.gethostname()
52 port = 9500
53 SocketServeur.bind(("127.0.0.1", port))
54
55 SocketServeur.listen(5)
56
57 contenu_clepublique = open("fichier_cle_publique.pem","rb").read()
58
59 contenu_cleprive = RSA.importKey(open("fichier_cle_prive.pem","rb").read())
60 # cree un objet a partir de la cle permetant de chiffre avec RSA
61 objet_cle_rsa_prive = PKCS1_OAEP.new(contenu_cleprive)
62 print("Lancement serveur")
63 while True:
64    ConnexionAUnClient, addrclient = SocketServeur.accept()
65    thread.start_new_thread(Traiter_Connexion,(ConnexionAUnClient,addrclient))
66
```

Listing 1. Code serveur

- La différence entre ce programme est celui vu dans la fin de la partie précédente, c'est que le client initialement génère et transmet une clé AES au serveur (celle-ci est chiffrée avant la transmission avec la clé publique du serveur). Par la suite, au lieu de chiffrer chaque message avec la clé publique du serveur, les messages sont encodés avec Base64 en premier temps et puis chiffrés avec la clé AES.
- Le serveur à son tour reçoit la clé et crée un objet de déchiffrement AES. Par la suite, chaque message reçu par le client est déchiffré par AES et puis décodé avec Base 64.

Question

Complétez le code Serveur (A FAIRE 1, 2 et 3) ainsi que le client (A FAIRE 1 et 2) pour que la communication fonctionne.