TP Cryptographie appliquée - Partie 3 Encodage Base 64

Table des matières

Introduction

Vous pouvez télécharger les codes dans cette série directement du lien suivant (*TP Crypto Partie 2 RSA.rar*):

https://drive.google.com/file/d/1BJdxfFYfDxRQ3RVyMamZWVxro3MZvJqT/view?usp=sharing

Avant de pourvoir entamer ce TP, vous devez avoir Anaconda 3 compatible avec votre système installé :

https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86_64.exe

https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86.exe

Encodage Base 64

⚠ *Attention*

Vous n'avez pas besoins de copier les codes à partir de la série. Chaque ressource (programme python ou fichier) mentionné dans la série est téléchargeable à partir du 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 1ere 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 des 6 bits est représenté 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 comme suite :

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 4, ainsi 2 octets sont ajoutés (3 -(4%3)). Le résultat est le suivant :

01000001010000100100001101000100000000000000000000

- Diviser la donnée à encodé en bloque 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 concaténé dans un bloque.
3. Le le bloque est ensuite divisé en octets de 8 bits pour récupérer le contenu originale.

Bibliothèque base64

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. Essayez de décoder la phrase suivante avec la méthode b64decode. Quel est le résultat ?
   QnJhdm8sIHZvdXMgYXZleiBkw6ljb2TDqSBsZSBtZXNzYWdlLg==
2. Ajoutez plusieurs caractère = à la fin du code précédant et puis essayer de décoder. Quel est l'effet sur le résultat ?
3. Est ce que l'encodage Base64 offre la priorité de diffusion ? Pourquoi ?

Communication crypté AES avec encodage base64

III

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

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')
# ecriture du contenu chiffre
fichier_sortie.write(contenu_chiffre)
# fermeture du fichier
fichier_sortie.close()

```

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'octet 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 le 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
```

Question 3

- 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

Communication Client-Serveur

IV

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)


```

```
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 série 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 publique du serveur, les messages sont encodés avec Base64 en premier temps et puis chiffré 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.