TP Cryptographie

TP 3: Cryptographie modern

Fonctions Hachage – AES – RSA

Installez le package python « *PyCryptodome* » avec la commande "pip install -U PyCryptodome". Veuillez-vous assurer que le package « *PyCrypto* » est désinstallé "pip uninstall PyCrypto".

1-Fonction de Hachage

Plusieurs algorithmes de hachage existent actuellement : MD5, SHA, CRC. La bibliothèque python standard inclut ces fonctions dans la bibliothèque *hashlib*. Ces algorithmes sont très similaires aux algorithmes de cryptographie AES et DES. Chaque algorithme produit une valeur de hachage de taille fixe quelle que soit la taille des données d'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. Les mots de passe des utilisateurs sont souvent enregistrés sous forme de hash au lieu de leurs valeurs claires. ci-dessous, un exemple d'utilisation des fonctions de hachage.

```
# importe les fonctions sha256, sha1 et crc32
from hashlib import sha256
from hashlib import sha1
from zlib import crc32

# crée un message encode en octets (le b au debut
# de la chaine de caracteres signifie byte)

message=b"Un message a transporter"

# calcule et affiche le hashage du message
# avec chaque algorithme


print("sha256 = ",sha256(message).hexdigest().upper())
print("sha1 =", sha1(message).hexdigest().upper())
print("crc32 =",hex(crc32(message))[2::].upper())
```

Travail demandé

- Écrivez un programme permettant de lire un fichier comme une suite d'octet (de préférence le fichier ne doit pas dépasser 1Mo). Concaténez les octets lus dans une variable appelée "data"
- Calculez le hash SHA256 et MD5 de data.
- Changez un seul caractère du message claire et comparez le résultat produit après modification avec le résultat du hashage avant modification.
- Est-ce que la propriété de diffusion est assurée par les algorithmes de hashage? Est ce que deux messages / fichiers différent peuvent avoir le même résultat de hashage ?

2-Chiffrement avec AES

La bibliothèque Crypto offre l'implémentation de plusieurs algorithmes de chiffrement comme AES et RSA. Pour commencer, il faudra générer une clé primaire aléatoire selon la taille de la clé supportée par l'algorithme. Ceci peut être assuré par la fonction suivante cle*_16_octet =os.urandom(16)*. Le code suivant montre comment chiffrer un contenu avec AES en utilisant la bibliothèque de cryptographie Crypto :

```
import os
from Crypto.Cipher import AES

# os.urandom(N) genere une sequence
# de N octets aleatoire
# cle de 16 octets (128 bits) aletoire
cle_16_octet =os.urandom(16)

#cree une instance AES avec une cle= "cle16 octets AES"
#la taille de la cle ==  (16octets)
objet_de_chiffrement=AES.new(cle_16_octet,  AES. MODE_ECB)

Message_claire=b"Message claire16"
print(Message_claire, len(Message_claire))

#chiffre le message claire
contenu_chiffre=objet_de_chiffrement.encrypt(Message_claire)
print(contenu_chiffre)

#Dechiffre le message chiffre
objet_de_chiffrement=AES.new(cle_16_octet,  AES. MODE_ECB)
Message_claire=objet_de_chiffrement.decrypt(contenu_chiffre)
print(Message_claire)
```

Important:

- Le texte à chiffré 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 de octets de bourrage à la fin du contenu, il existe plusieurs méthodes :
    - Ajout de la taille du message original sans bourrage avant le contenu chiffré.
    - Ajout du nombre d'octets de bourrage avant le contenu chiffré.
- Utiliser un caractère absent du contenu claire pour les octets de bourrage pour marquer la fin lors du déchiffrement.
- Lors de l'initialisation de l'algorithme, AES automatiquement obtient la taille de la clé (128 bits, 192 bits ou 256 bits) pour déterminer la taille du bloque.

Travail demandé

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

- Écrivez un programme permettant de lire le contenu d'un fichier sous forme de séquence d'octets.
- Le programme doit calculer la taille en nombre d'octets du contenu. 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*.
- Ajoutez les octets de bourrage selon la valeur obtenue.
- Générez une clé de 16 octets et sauvegarder la clé dans fichier *key.bin* afin de l'utiliser lors de la question de déchiffrement.
- Chiffrez le contenu avec AES.
- Ajoutez un octet au début du contenu chiffré contenant la valeur de bourrage.
- Enregistrez le résultat dans un fichier.

3-Chiffrement avec RSA

Génération de clé RSA

RSA est un algorithme de chiffrement asymétrique. Celui-ci nécessite la génération d'une clé publique et d'une clé privée. La clé publique est sauvegardée dans
« *fichier_cle_publique.pem* »
La clé privée est sauvegardée dans
«*fichier_cle_prive.pem*».
Le programme suivant illustre comment génère des clés RSA.

```
# importe le module RSA
from Crypto.PublicKey import RSA

# genere une cle publique/prive avec modulos (N) de 2048 bits
cle= RSA.generate(2048)
```

```
# recupere la cle prive
cle_prive = cle.exportKey()

#ouvre un fichier en mode d'ecriture en octets
Fichier = open("fichier_cle_prive.pem", "wb")

# ecrit le contenu de la cle prive sur fichier
Fichier.write(cle_prive)
# ferme le fichier
Fichier.close()

# recupere la cle publique
cle_publique = cle.publickey().exportKey()


# ecrit le contenu de la cle publique sur fichier
Fichier = open("fichier_cle_publique.pem", "wb")
Fichier.write(cle_publique)
Fichier.close()

#imprime la cle publique
print(cle_publique)

#imprime la cle RSA
print(cle_prive)
```

Chiffrement et déchiffrement avec RSA

Le programme suivant permet d'importer les clés publique et privée ainsi que le chiffrement avec clé publique et déchiffrement avec clé privée. Un objet permettant de chiffrer avec la clé publique est affecté à objet_cle_rsa. Un objet permettant de chiffrer avec la clé privée est affecté à objet_cle_rsa_prive.

```
import Crypto
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))
```

A base de cette code, créez un programme permettant de chiffrer/déchiffrer avec RSA .

Communication avec serveur sécurisée

Pour commencer, vous devez générer les deux fichiers contenant la clé publique et la clé privée avec le programme expliqué dans la partie 1. Les deux programmes représentent un code du serveur multi-thread d'un client. Ces derniers doivent entre places dans le même répertoire que les fichiers des clés publique et privé.

- Le serveur multi-thread transmet la clé publique au client initialement (contenu_clepublique) ;
- Par la suite les messages reçus du clients doivent être déchiffres 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 resulta du dechiffrement a MessageRec
          # (A FAIRE 2) ajoutez l'instruction dans cette ligne



          if MessageRec==b"Fin":
                  break

          print("Client" ,adresse_client," a dit :",MessageRec.decode())
   except:
      print("Deconnexion")
   print("Deconnexion de :",adresse_client)
   try:
      connexion_avec_client.close()
   except:
      pass

SocketServeur = socket.socket()
host = socket.gethostname()
port = 9500
SocketServeur.bind(("127.0.0.1", port))

SocketServeur.listen(5)

contenu_clepublique = open("fichier_cle_publique.pem","rb").read()

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)
print("Lancement serveur")
while True:
   ConnexionAUnClient, addrclient = SocketServeur.accept()
   thread.start_new_thread(Traiter_Connexion,(ConnexionAUnClient,addrclient))
```

- Après la connexion au serveur, le client 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 chaque message saisie par le client, celui-ci est chiffré avec la clé publique (objet_cle_rsa_publique) et transmet le résultat (resultat_chiffre) ;

```
import socket
from Crypto.PublicKey import RSA
from Crypto.Cipher import  PKCS1_OAEP

SocketClient = socket.socket()
host = socket.gethostname()
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

# Importez la cle a partir du contenu recu Cle_publique
# Utilisez la methode importKey de RSA pour importer la cle
# Mettez le resultat dans clepublique
# Creez un objet objet_cle_rsa_publique a partir de clepublique (objet de cle
publique)
# Utilisez PKCS1_OAEP.new pour faire ceci
# (A FAIRE 1) ajoutez deux instructions dans les deux lignes suivantes


print("Lancement serveur")
while True:
   MessageATransmettre=input().encode()
   # Chiffre le message lu du clavier (MessageATransmettre)
   # Utilisez la methode encrypt de objet_cle_rsa_publique
   # Mettez le resultat dans resultat_chiffre
   # (A FAIRE 2) ajoutez l'instruction dans cette ligne



   SocketClient.send(resultat_chiffre)
   if(MessageATransmettre==b"Fin"):
      break
      print( "Deconnexion de :",addrclient)
ConnexionAUnClient.close()
```

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 FAIRE 1). Une instruction suffirais pour ceci ;

2. Programme Serveur : Déchiffrer le contenu reçu du client [MessageRec] avec la clé privée [objet_cle_rsa_prive](A FAIRE 2). Une instruction suffirais pour ceci ;
3. 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).
4. Programme Client : Créer un objet de la clé publique à partir de clepublique, et mettre le résultat dans l'objet objet_cle_rsa_publique.
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) ;