TP Cryptographie appliquée - Partie 1

Bambrik Ilyas

Table des matières

Introduction

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

Fonction de hashage

- 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 actuelement : 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 valeur de 160 bits (20 octets) et MD5 avec 128 bits.

Remarque : Les mots de passe des utilisateurs sont souvent enregistrés sous forme de hash au lieu de leurs valeurs claires.

```python
# 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())
```

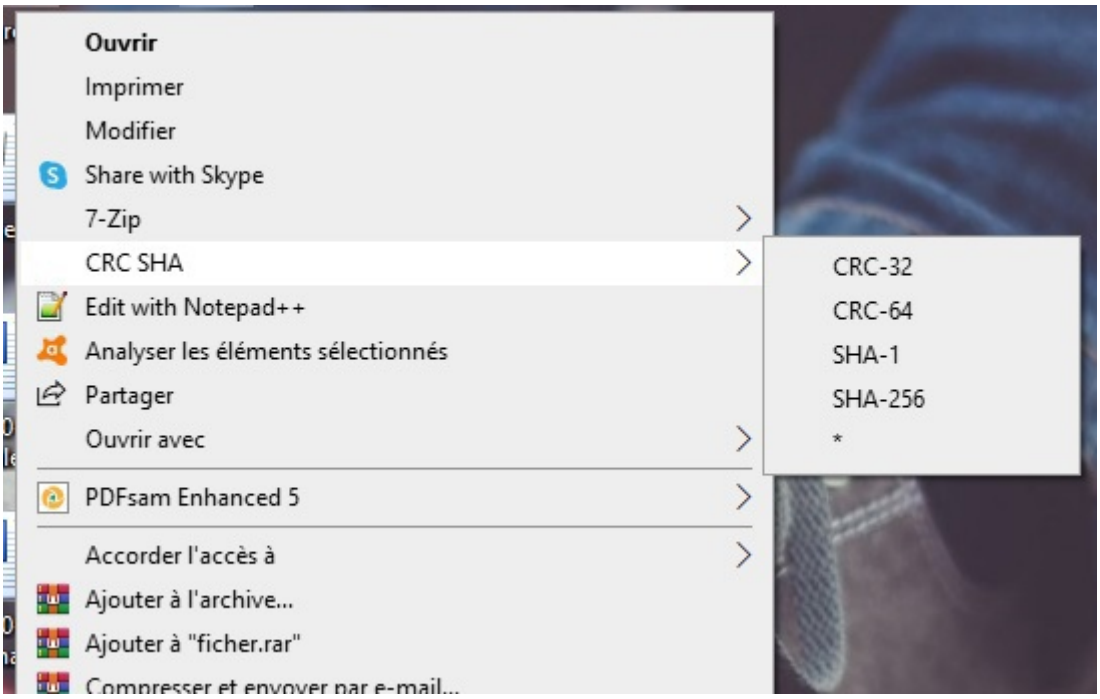
La sortie du hashage de "Un message a transporter" est (l'affichage est en hexadécimale):

```
sha256 = 16F71232D915F3533D6510E032EB1C4BFEF2AF39E3EB44A0B58B9C44D4A65361

sha1 = 2B3A855FBA7F4A19090DCB61AC5C7011E7D12AFC

crc32 = CB554B3E
```

Afin de vérifier bien que la valeur 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 hash 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'octet (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 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 message claire et comparez le résultat produit après modification avec le résultat du hashage avant modification.
5. *Est ce que la propriété de diffusion est assurée par les algorithmes de hashage? 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'octet data vide, 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()
```

Chiffrement avec AES

- La bibliothèque Crypto inclue dans Anaconda offre l'implémentation de plusieurs algorithme 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. 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 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 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 :
    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 pour 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 :

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

Résultat :

```
b'Message claire16'
```

Question 1

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

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*. Sauvegarder la clé dans 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.

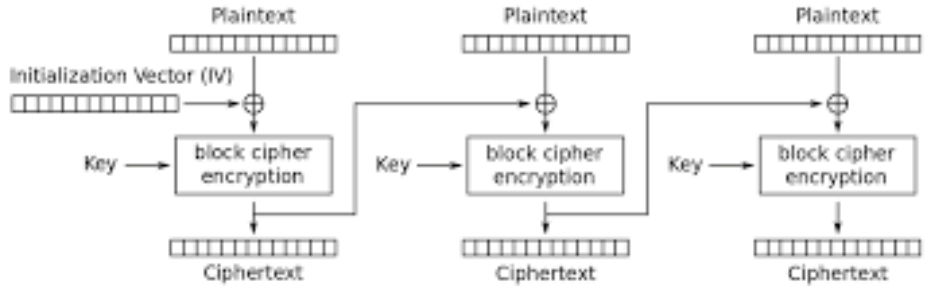
Vecteur d'Initialisation (IV) et Cipher-Block Chaining (CBC)

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 etre 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 :

```
1
2 #importe AES
3 from Crypto.Cipher import AES
4 #cree une instance AES avec une cle= "cle16 octets AES"
5 #l'argument AES.MODE_CBC indique que le chiffrement est
6 # en mode CBC
7
8 #le vecteur d'initialisation = "vecteur d'init16"
9
10 objet_de_chiffrement=AES.new("cle16 octets AES",AES.MODE_CBC,"vecteur d'init16")
11 Message_claire="Message claire16"
12 #chiffre le message claire
13 contenu_chiffre=objet_de_chiffrement.encrypt(Message_claire)
14 print(contenu_chiffre)
15
16 #dechiffrement avec CBC
17 objet_de_dechiffrement=AES.new("cle16 octets AES",AES.MODE_CBC,"vecteur d'init16"
   )
18
19 text_claire=objet_de_dechiffrement.decrypt(contenu_chiffre)
20 print(text_claire)
21
```

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 supérieur 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*)
2. Chiffrez le message avec AES 128bits sans CBC (comme dans le premier exemple AES, page 7 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, page 9 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 ?