Compte-rendu module Cryptographie Groupe3

BA Demba, GUENFICI Rayane, SASSIKUMAR Suban, ZIHOUNE Bilal, MENDES Fredy

1. Introduction

Dans le cadre de notre Projet SAE du Semestre 3, nous devions réaliser un module de Cryptographie implémentant le cryptage RC4 et WEP. Ainsi, les utilisateurs peuvent chiffrer un texte avec une clé (tous deux en ASCII) et retourner le texte chiffré en hexadécimal. Vous pourrez trouver notre code à l'adresse suivante : https://gitlab.com/sae_3_01/sae_3_01/ .

2. RC4

Notre code (chemin : sae_3_01/src/L4/python_module2/rc4.py) implémentant l'algorithme RC4 contient quatre parties :

- La conversion de la clé et du texte à chiffrer en ASCII :

```
key = [ord(c) for c in key] # valeur ASCII des lettres de la clé
    if action == "c" : #chiffrement
        message = [ord(c) for c in message] # valeur ASCII des lettres du message
    else :
        message = hexa_to_ten(message) # convertion des nombres hexadécimaux en décimaux
```

- La fonction hexa_to_ten :

```
def hexa_to_ten(str):
    """
    Convertit une chaîne de charactères de nombres héxadéciamaux
    (base 16) liste de nombres décimaux (base 10)

    Entrée :
            str (str) : Nombres héxadécimaux

    Sortie :
            (list) : Nombres décimaux
    """
    liste = str.split(" ")
    return [int('0x' + cara , 0) for cara in liste]
```

- L'initialisation de la suite chiffrante :

```
    suite = list(range(256))
    j = 0
    for i in range(256):
```

```
        j = (j + suite[i] + key[i % len(key)]) % 256
        suite[i], suite[j] = suite[j], suite[i]
```

- Génération de la permutation:

```
    # Appliquer l'algorithme RC4 au message (cf KSA dans le cours)
    result = []
    i = j = 0
    for lettre in message:
        i = (i + 1) % 256
        j = (j + suite[i]) % 256
        suite[i], suite[j] = suite[j], suite[i]
        result.append(lettre ^ suite[(suite[i] + suite[j]) % 256])
        # ^ applique l'opérateur logique xor
```

- La conversion en héxadécimal (en cas de chiffrement)

```
if action == "c":
    return ten_to_hexa(result)
```

```
def ten_to_hexa(liste):
    """Convertit une liste de nombres décimaux (base 10) en
    une chaîne de charactères de nombres héxadécimaux (base 16)

    Entrée :
            (list) : Nombres décimaux
    Sortie :
            str (str) : Nombres héxadécimaux

    """
    liste = [hex(e)[2:].upper() if len(str(hex(e)))>3 else "0" + hex(e)[2:].upper() for e in liste]
    return " ".join(liste)
```

- La conversion en lettre (si déchiffrement):

```
return ''.join([chr(e) for e in result])
```

3. WEP

D'après l'article de Stonic and Bogdanovic *RC4 stream cipher and possible attacks on WEP*, le protocole WEP chiffre les packets en utilisant la clé fournie par l'utilisateur et un vecteur d'initialisation (IV) généré de manière aléatoire (avec l'algorithme RC4). Le IV occupant 3 bit, sa valeur peut varier entre 00 00 00 et FF FF FF (en hexadécimal), soit entre 0 et 16777215. La clef de chiffrement devient la concatenation de IV et de la clé entrée.

Le chemin du code : sae_3_01/src/L4/python_module2/wep.py

Génération du vecteur d'initialisation:

```
def IV():
    """
    Génere une clé àléatoire (aussi appelée IV pour Initialization Vector) de 24 bits.

    Entrée : None
    Retour : list : 3 entiers héxadécimaux (1 héxadécimal = 8 bit)
    """
```

```
    iv = randint(0,16777215) # Génére un entier aléatoire ent 0 et 16777215
                             # 16777215 car 16777215 = FF FF FF
    nb = str(hex(iv)[2:]).upper() # convertion de l'entier en haxadécimal puis en string pour
                                  # supprimer les '0x' ajoutés par python
    while len(nb) < 6:            # remplissage de 0 : si iv = FF, donne 0000FF
        nb = "0" + nb
    nb = [nb[i:i+2] for i in range(0,len(nb),2)] # Regroupement par 2 (par bit)
    nb = [int('0x' + e,0) for e in nb] # convertion de chaque bit en hexadécimal
    return nb
```

Les modifications par rapport aux R sont représentés par des doubles commentaires (##):

```
 if action == "c" : # Chiffrement
        message = [ord(c) for c in message] # valeur ASCII des lettres du message
        iv = IV() ## géneration de la clé aléatoire

 else :# Déchiffrement
        message = hexa_to_ten(message) # Convertion hexadécimal
        iv = message[:3] ## clé aléatoire
        message = message[3:] ## reste du message


# Algorithme RC4 (voir suite chiffrante et permutation ci-dessus)
[...]

 if action == "c": # Chiffrement
        return ten_to_hexa(iv + result) ## Concatenation IV (3 bits) et message chiffré
 else : #si déchiffrement
        return ''.join([chr(e) for e in result])

'''
```