

Compte-rendu module Cryptographie Groupe3

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

1. Introduction

2. RC4

3. WEP

1. Introduction Dans le cadre de notre Projet SAE du Semestre 3, nous devons 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 obtenir le texte chiffré en hexadécimal. Vous pourrez trouver notre code à l'adresse suivante : https://bitbucket.org/bademba09/sae_3_01.git. (branche master). Pour l'obtenir, il vous faut créer un nouveau dossier puis exécuter la commande suivante : `git clone https://bitbucket.org/bademba09/sae_3_01.git`

Par ailleurs, si vous voulez tester notre code, vous pouvez le faire directement en terminal. Exemple : Chiffrement : `python rc4.py c Plaintext Key` Dechiffrement : `python rc4.py d 'BB F3 16 E8 D9 40 AF 0A D3' Key`

Ou bien vous pouvez accéder à notre Raspberry PI sur les machine de l'IUT à l'adresse : 192.168.1.163, créer votre compte et accéder aux modules.

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

1. 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 : #dechiffrement
    message = hexa_to_ten(message) # conversion des du hexadécimal en décimaux (voir hexa_to_ten())
```

- La fonction `hexa_to_ten` :

```
def hexa_to_ten(str):
    """Convertit un nombre hexadécimal (en str) en liste de nombres décimaux (base 10)

    Entrée :
        str (str) : Nombre hexadécimal

    Sortie :
        (list) : Nombres décimaux
    """
```

```

liste = list(str)

# regroupe les caractères par deux ex : A0A0A0 -> [[A],[0," "],[A],[0," "],[A],[0]]
liste = [[liste[e]," "] if e%2 == 1 and e != len(liste) -1 \
          else [liste[e]] for e in range(len(liste))]

#applatit la liste ex : [[A],[0," "],[A],[0," "],[A],[0]] -> [A,0," ",A,0," ",A,0,]
liste = sum(liste, [])

#regroupe les elements dans une str ex : [A,0," ",A,0," ",A,0,] -> "AO AO AO"
liste = "".join(liste)

#sépare les éléments : "AO AO AO" -> [AO,AO,AO]
liste = liste.split(" ")

return [int('0x' + cara , 0) for cara in liste] # [10,10,10]

```

L'intérêt de cette fonction est que, pour appliquer la permutation (voir 2.3), nous devons convertir chaque octet du message hexadécimal en décimal car la clé suite chiffrente est en décimal. Ce n'est donc pas une simple conversion d'hexadécimal en décimal. Aussi, $(A0A0A0) ^{(_16)} = (10526880) ^{(_10)}$

2. L'initialisation de la suite chiffrente :

```

#Pseudo Random Generation Algoritm
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]

```

3. 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

```

4. Output:

- La conversion en hexadécimal (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 un nombre hexadécimal

```

```

Entrée :
    (list) : Nombres décimaux
Sortie :
    str (str) : Nombre hexadécimal
Ex : [10, 10, 10] -> A0 A0 A0
"""
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 octets, sa valeur peut varier entre 00 00 00 et FF FF FF (en hexadécimal), soit entre 0 et 16777215. La clé 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():
    """
    Génère une clé aléatoire (aussi appelée IV pour Initialization Vector) de 3 octets (24 bits)
    et les retourne sous forme décimale.

    Entrée : None
    Retour : list : 3 entiers décimaux qui sont la conversion de chaque octet
    EX ; IV = 16000000 ; IV = F42400 (base 16)
    retour = [F4 , 24 , 00] -> [244 , 36, 00]
    """

    iv = randint(0,16777215) # Génère un entier aléatoire entre 0 et 16777215
                           # 16777215 car 16777215 = FF FF FF
    nb = str(hex(iv)[2:]).upper() # conversion de l'entier en hexadécimal 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 octet)
    nb = [int('0x' + e,0) for e in nb] # conversion de chaque octet en décimal
    return nb

```

Les modifications par rapport aux RC4 sont représentées par des doubles commentaires (##):

```

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

else :# Déchiffrement
    message = hexa_to_ten(message) # Conversion hexadécimal

```

```

        iv = message[:3] ## extrait IV (3 octet)
        message = message[3:] ## reste du message

# Algorithme RC4 (voir suite chiffrente 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])

'''

```