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 obtenir le texte chiffré en hexadécimal.

Vous pourrez trouver notre code sur la boite de dépot de la SAE sur ECampus.

Pour le tester, il vous faudra ouvrir un terminal et exécuter la commande *python [nom_fichier] [action] [message] [clé]*

Exemple :

Chiffrement : python rc4.py c Plaintext Key

Dechiffrement : python rc4.py d 'BB F3 16 E8 D9 40 AF 0A D3' Key

A noter que, pour le déchiffrement, il vous faut entrer le message hexadécimal avec des guillemets ou sans aucun espace ('BB F3 16 E8 D9 40 AF 0A D3' ou BBF316E8D940AF0AD3)

Vous pouvez également accéder à notre Raspberry PI sur les machines de l'IUT à l'adresse : 192.168.1.163, créer votre compte et accéder aux modules.

2. RC4 Notre code (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) # convertion des octets 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
    """
    if " " in str:
        str = str.replace(" ","") #  Supprime les espaces
    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,] -> "A0 A0 A0"
    liste = "".join(liste)

    #  sépare les éléments : "A0 A0 A0" -> [A0,A0,A0]
    liste = liste.split(" ")

    return [int('0x' + cara , 0) for cara in liste] # [160,160,160]
```

L'intéret de cette fonction est que, pour appliquer la permutation (voir 2.3), nous devons convertir chaque octet du message hexadécimal en décimal car la suite chiffrante est en décimal. Ce n'est donc pas une simple conversion d'un nombre hexadécimal en décimal.

2. L'initialisation de la suite chiffrante :

```
    #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 (octets) 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 (avec l'algorithme RC4) les paquets en utilisant la clé fournie par l'utilisateur et un vecteur d'initialisation (IV) généré de manière aléatoire. Le IV occupant 3 bit, sa valeur peut varier entre 00 00 00 et FF FF FF (en hexadécimal), soit entre 0 et 16777215. Il y a donc 16777216 possibilités de chiffrement d'un paquets avec une seule clé contre 1 avec le RC4

La clef de chiffrement devient la concatenation de IV (la clé publique) et de la clé entrée (la clé privée).

Pour le déchiffrement, il suffit de récupérer les 3 premiers octets du message chiffré et de les concatener dans le sens : IV + clé.

Nom du fichier : wep.py

Génération du vecteur d'initialisation:

```
def IV():
    """
    Génere une clé àléatoire (aussi appelée IV pour Initialization Vector) de 3 octets (24 bits)
    et les retournes sous forme décimale.

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

    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 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] # convertion de chaque octet en décimal
    return nb
```

Les modifications par rapport aux RC4 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] ## extrait IV (3 octet)
        message = message[3:] ## reste du message

 key = [ord(c) for c in key] ## Conversion lettres -> code ASCII
 key = iv + key ## Concatenation IV et clé


# 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])
```