Compte Rendu

R3.09-Cryptographie

TP3+4

Table des matières

A.Recherche Bibliothèque :

Bibliothèques disponibles :

- En Python : hashlib, cryptography, pycryptodome
- En Java : java.security.MessageDigest, BouncyCastle
- En Rust : ring, rust-crypto
-

J'ai choisi d'utiliser Python . Pour la bibliothèque j'ai choisi hashlib pour les raisons suivantes :

- Elle est déjà installée par défaut dans les versions récentes de Python.
- La syntaxe est simple en particulier pour le hash SHA-256 ou SHA-512.
- Elle est facile à comprendre et rapide à utiliser.
- La documentation est facilement trouvable sur internet.

B. Génération du mot de passe :

Cas taille fixe 8 caractères :

```python
#Hash de la concatenation de deux chaines de caractères
chaineconat = mdpmaitre + tag
chaine = hashlib.sha256()
chaine.update(chaineconat.encode('utf-8'))

#Découpé le hash en 8 parties de 8 caractères
chaine = chaine.hexdigest()
tab = [0]*8
for i in range(8):
    tab[i] = chaine[i*8:(i+1)*8]
```

J'ai décidé d'utiliser la fonction de Hash SHA256 ce qui me retournera une chaîne de 256 bits soit 64 caractères ce qui est un multiples de 8 , donc j'ai décidé de découper ma chaîne de 64 caractère en 8 chaînes de 8 caractères.

```python
#Traduction de chaque partie en caractère
resultat = ""
for i in range(8):
    #addition des valeurs de chaque bloc
    somme = 0
    for j in range(8):
        somme += ord(tab[i][j])
    #traduction en caractère normaux commme spéciaux
    resultat += chr(somme%94 + 33)
print(resultat)
```

Après pour chaque chaînes je la parcours et converti chaque lettre en entier que j'additionne et juste après je fait le modulo de mon résultat ce qui peut me retourné un caractère normal comme spécial

H("lemotdepasse","Unilim")

E}R,|?M{

Cas taille variable :

```python
#Hash de la concatenation de deux chaines de caractères
chaineconat = mdpmaitre + tag
chaine = hashlib.sha512()
chaine.update(chaineconat.encode('utf-8'))

#Découpé le hash en taille parties
chaine = chaine.hexdigest()
tab = [0]*taille
decoupe = len(chaine)//taille
for i in range(taille):
    tab[i] = chaine[i*decoupe:(i+1)*decoupe]
#Traduction de chaque partie en caractère
resultat = ""
for i in range(taille):
    #addition des valeurs de chaque bloc
    somme = 0
    for j in range(decoupe):
        somme += ord(tab[i][j])**taille+decoupe
    #traduction en caractère normaux commme spéciaux
    resultat += chr(somme%94 + 33)
return(resultat)
```

Dans le cas de la taille variable de mot de passe j'ai préféré passer sur du SHA 512 ce qui me permet d'avoir un mot de passe de taille plus grande que si j'utilisai SHA 256 car la chaîne de caractères fait 512bit au lieu de 256bit , j'ai décidé de la découper dans un nombre égale de part

Après pour chaque chaînes je la parcours et converti chaque lettre en entier que j'additionne je le met à la puissance de la taille du mot de passe et j'ajoute la taille des bloc (ce qui me permet de faire comme un salt) et juste après je fait le modulo de mon résultat ce qui peut me retourné un caractère normal comme spécial

print(H("lemotdepasse","Unilim",12))

4N{kUy}y'|%4

C. Mot de passe maître

```python
try:
    with open('mpwd.txt', 'r') as fichier:
        pass
except FileNotFoundError:
    with open('mpwd.txt', 'w') as fichier:
        pass


# Ouvrir et lire le fichier
with open('mpwd.txt', 'r') as fichier:
    contenu = fichier.read()

    if not contenu:  # Si le contenu est vide
        Changer_MDP()
```

Système qui vérifie si le fichier existe sinon il le crée , puis après il va ouvrir et lire le fichier et si le contenu est vide alors on va lancer la fonction Changer_MDP() qui permet de changer le mot de passe

```python
#Menu
print("1. Changer le mot de passe maître")
print("2. Générer un mot de passe")
print("3. Quitter")
choix = input("Votre choix : ")
if choix == "1":
    Changer_MDP()
    Exercice3()

elif choix == "2":
    # Lire le mot de passe maître
    mdp = Lire_MDP()
    # Demander le tag et la taille
    tag = input("Saisir le tag : ")
    taille = int(input("Saisir la taille : "))
    # Appeler la fonction H
    print(Exercice2.H(mdp, tag, taille))
    Exercice3()

elif choix == "3":
    print("Au revoir !")
else:
    print("Choix invalide.")
    Exercice3()
```

Le menu qui va nous permettre de changer le mot de passe maître ou de générer un nouveau mot de passe à l'aide d'un tag et d'une taille en appelant la fonction de l'exercice 2

```python
def Changer_MDP():
    with open('mpwd.txt', 'w') as fichier:
        mdp = input("Saisir le mot de passe maître : ")
        # Ecrire le mot de passe dans le fichier
        fichier.write(mdp)
        print("Le mot de passe a été changé.")

def Lire_MDP()-> str:
    with open('mpwd.txt', 'r') as fichier:
        mdp = fichier.read()
        return mdp
```

Les deux fonctions qui permette de changer et de lire le mot de passe

Attaque :

Pour notre attaque on va faire une attaque brute force qui se base sur le phénomène de collision , qui se produit car plusieurs mot de passe peuvent théoriquement le même hash car on a un nombre de mot de passe infini et une sortie du hash de taille limité , c'est l'une des seule faille que l'on peut utiliser car on peut pas renverser le processus du hash car un hash doit être one-wayness.

```python
def BruteForceMultiTag(MdpMaitre1: str, MdpMaitre2: str,MdpMaitre3: str,Tag1: str , Tag2:str, Tag3:str, taille: int) -> str:
    """…
    # Définit les caractères possibles (ASCII entre 33 et 126)
    caracteres = [chr(i) for i in range(33, 127)]

    # Génère toutes les combinaisons possibles pour un mot de passe de longueur "taille"
    for combinaison in itertools.product(caracteres, repeat=10):
        mdp = ''.join(combinaison)

        # Vérifie si le mot de passe généré correspond aux trois tags ce qui permet de trouver la clé exacte
        if E2.H(mdp,Tag1 , taille) == MdpMaitre1 and E2.H(mdp,Tag2 , taille) == MdpMaitre2 and E2.H(mdp,Tag3 , taille) == MdpMaitre3:
            return mdp
```

Fonction de création de mot de passe:

X → H(X+tag) → Y

Donc on connaît seulement :

tag1 → Y1

tag2 → Y2

tag3 → Y3

et H()

Donc ce que l'on va faire c'est générer des mots de passe de taille 10 comme demander et on va regarder que quand on la passe dans la fonction de génération de mot de passe avec les 3 tag si les mots de passe généré correspond et normalement quand ça correspond on obtient le mot de passe maître ou a une collision qui correspond aux 3 tags .

Essaie d’attaque

On va essayer de faire une attaque sur un mot de passe avec 3 tag :

Le mot de passe maître est : lemotdepas

Les tag sont : Unilim , Facebook et X

Taille mot de passe	1	2	3
Nombre essaye	Environ 542 000	Arrêt à 185 534 000 mais pas trouver	Pas essaie car presque impossible

Pour trouver une approximation du nombre d’essaie qu’il faut on peut utiliser la « formule »

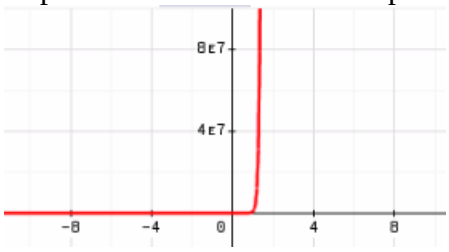
(nombre de lettres disponibles^taille mot de passe)^nombre de tag

pour 1 caractères :
(94^1)^3 = 830 584

pour 2 caractères :
(94^2)^3 = 6,8986e11

pour 3 caractères :
(94^3)³ = 5,7299e17

Représentation de la fonction pour trois tag sous forme d’un graph :



et sous forme d’un tableau de pas 1 :

x	f(x)
0	1
1	830 584
2	6.898697811e11
3	5.729948022e17
4	4.759203148e23
5	3.952917988e29
6	3.283230434e35

Conclusion :

En conclusion le générateur de mot de passe que l’on a élaborer durant ce TP , peut être considérer comme efficace car la difficulté pour ne serrai-ce trouver une collision à l’aide de trois tag est d’une complexité exponentielle donc pour trouver un mot de passe maître en entier c’est de l’ordre du presque impossible .

Réflexion sur l’utilisation d’un générateur de mot de passe :

L’utilisation d’un générateur de mot de passe basé sur SHA512 est efficace mais pas infaillible à des attaque par dictionnaire donc l’utilisation de mot de passe maître trop simple peut être une faille de notre solution c’est pour cela qui faut utiliser un mot de passe maître qui soit complexe ou long et il ne faut pas qu’il soit stocké en claire sur votre machine donc qu’il soit chiffré de manière sécurisé donc avec AES 256 , SHA512 par exemple et pas par MD-5 ou SHA-1 qui sont obselète .