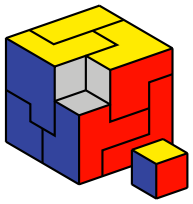
Prologin



Concours National d'Informatique

Correction des challenges de pré-lancement

Correction des challenges de pré-lancement

Kilian Guillaume
Victor Collod

4 Janvier 2019

Table des matières

1 Challenge 1 **2**

2 Challenge 2 **4**

3 Challenge 3 **6**
3.1 Approche 1 6
3.2 Approche 2 8
3.3 Approche 3 8

1 Challenge 1

Énoncé Notre radioastronome Haruhi a enregistré et décodé un étrange message. Elle a obtenu ce court programme. Peux-tu l'aider à trouver ce qu'il recèle ?

Résolution En l'état, le programme n'est pas lisible. En effet, le programme est au format `.jar`, c'est à dire qu'il a déjà été traduit depuis le langage texte java, lisible par un humain, au langage que comprend la JVM [1], un autre langage pratiquement équivalent mais lisible plus efficacement par une machine.

La solution la plus simple est certainement d'effectuer la traduction inverse. Il existe une floppée d'outils, dont certains sont disponibles en ligne [2].

```
import java.io.PrintStream;

public class Main {
    public static void main(String[] args) {
        String str_a = "jx5PBuFnQiESQag8c3KmhvFlwnaMGYrJeW";

        char[] str_b = {
            '&', 29, 21, '6', '.', 20, '!', 'N', '4', 26, '1', 's', 'k', 'A', '+',
            'T', 4, '^', '=', '=', 1, 3, 18, 31, '#', '\n', 'X', '%', 's', ' ', '5',
            29, 20, 'o'
        };

        if ((args.length != 1)
            || (args[0].length() != str_a.length())) {
            System.out.println("Vous ne nous avez pas encore compris, recommencez.");
            return;
        }

        for (int i = 0; i < str_a.length(); i++)
            if ((args[0].charAt(i) ^ str_a.charAt(i)) != str_b[i]) {
                System.out.println("Vous ne nous avez pas encore compris, recommencez.");
                return;
            }

        System.out.println("Vous avez passé cette étape, nous nous reverrons.");
    }
}
```

Le code utilise l'opérateur ou exclusif (xor) pour valider les caractères un à un ($entre \oplus char_a = char_b$). Hors, cet opérateur permet l'égalité suivante : $char_a \oplus char_b = solution$

Ce code peut être modifié de la sorte pour afficher le mot de passe attendu :

```
public class Main {
    public static void main(String[] args) {
        String str_a = "jx5PBuFnQiESQag8c3KmhvFlwnaMGYrJeW";

        char[] str_b = {
            '&', 29, 21, '6', '.', 20, '!', 'N', '4', 26, '1', 's', 'k', 'A', '+',
            'T', 4, '^', '=', '=', 1, 3, 18, 31, '#', '\n', 'X', '%', 's', ' ', '5',
            29, 20, 'o'
```

1. https://fr.wikipedia.org/wiki/Machine_virtuelle_Java
2. http://www.javadecompilers.com/ permet de choisir parmis un certain nombre de décompilateurs

```
        };

        for (int i = 0; i < str_a.length(); i++)
            System.out.print((char)(str_a.charAt(i) ^ str_b[i]));

        // on termine la ligne
        System.out.println();
    }
}
```

On peut le compiler et le lancer comme suit :

```
$ javac Main.java
$ java Main
Le flag est : LlgmvPiuTsTd9h4yGWq8
```

Correction proposée par Victor Collod (multun).

2 Challenge 2

Énoncé Un portail web avec un champ pour se connecter est fourni.

Résolution Cette page web vérifie un mot de passe côté navigateur.

Voici l'algorithme utilisé :

— lorsque l'utilisateur appuie sur vérifier, le mot de passe donné est découpé en blocs de 4 caractères. Si la taille du mot de passe n'est pas un multiple de 4 caractères, des caractères nul sont rajoutés pour combler l'espace manquant

— pour chaque bloc de 4 caractères, on applique l'algorithme de hash sha256. Si le hash obtenu n'est pas le même que celui des 4 caractères du vrai mot de passe, à la même position, le mot de passe est faux.

Cet algorithme a un problème : Il découpe le mot de passe en blocs de 4 caractères. Il est ainsi possible de calculer les hashs de tous les mots de passe de 4 caractères, de les comparer aux valeurs connues, et d'en déduire des bouts du mot de passe.

Il y a au total 256^4 mots de passe de 4 caractères, soit 4 294 967 296 hashs à calculer.

Cependant, il est plausible que seul des caractères ascii soient utilisés. Explorer d'abord cette éventualité réduit le nombre de possibilités à $95^4+95^3+95^2+95$ [3] soit 82 317 120 hashs, ce qui est beaucoup plus raisonnable.

Nous allons donc générer toutes les combinaisons possibles, puis les comparer avec les hash stockés dans le code javascript afin de déterminer lesquelles sont valides.

Une solution possible en Python :

```python
import hashlib
from itertools import product, chain

# hashs extrait de l'array "sha256s" de la page html, converti en hexadecimal
blocs = [
    "58157cc492317721ec8dd734af2ed8fa2cf008085a00b292e8c2b4a13de6a3ff",
    "2766a545653c3e96650c1bbfee457d4ff11770eab5f6bf4f208a7436c240b50d",
    "2a647f5ce01a55201919afc3f7928ee12cd792cf2c14b52469951bb79cacc9c5",
    "8b31abac1296ea7c523ac8842e305bb9788eb56ba091c4f650aba29cc745e574",
    "053a9cc979d0d9766c3386c1641a3b5483342901118ae37fc633ae667daed562",
    "ddcddbe4434dae245ae520c1f4560403b08420cab54992566e92c9374f934481",
    "a4d4a38923939e7ed99bc3f8ae84459a6de890272b7fdefad2231e2c65f92f0b",
    "e4feb3d1276abc0033c89938fbe29cfec58c5bae4b0f8c3790d1b6b8b27956e5",
    "f5a478197a53ed67ee56260be35a5705cebe2bfbd08c6fd2583e91f3dacf39e6"
]

# on convertit la liste en dictionnaire, pour un accès plus rapide
blocs = {bloc: index_bloc for index_bloc, bloc in enumerate(blocs)}

print("En cours...")

solution = {}

def mots_possibles(taille_max):
    alphabet = list(range(32, 127))
```

3. 95 caractères possibles sur des mots de 4, 3, 2 et 1 caractères

```
    for taille in range(1, taille_max + 1):
        rembourage = (0,) * (taille_max - taille)
        for mot in product(alphabet, repeat=taille):
            yield bytes(chain(mot, rembourage))

for mot in mots_possibles(4):
    hasher = hashlib.sha256()
    hasher.update(mot)
    hash_mot = hasher.hexdigest()
    if hash_mot in blocs:
        index_bloc = blocs[hash_mot]
        bloc = mot.decode()
        print("bloc {} est {}".format(index_bloc, bloc))
        solution[index_bloc] = bloc
        if len(solution) == len(blocs):
            break

# on remet les blocs dans l'ordre
print(''.join(solution[i] for i in range(len(blocs))))
```

Sur mon ordinateur, le mot de passe est entièrement trouvé en 1min et 55s [4]
La sortie du programme est la suivante :

```
En cours...
block 8 est r3
block 4 est 0F3G
block 5 est 6lTg
block 3 est : 8v
block 0 est Le f
block 7 est e8fW
block 2 est est
block 1 est lag
block 6 est x7SF
Le flag est : 8v0F3G6lTgx7SFe8fWr3
```

À noter qu'il est possible d'effectuer ces calculs en parallèle. [5]. Une telle approche divise par le nombre de fils d'exécution le temps de calcul [6].

Correction proposée par Kilian Guillaume (CaféHaine).

Correction proposée par Victor Collod (multun).

4. ce temps peut énormément varier selon l'ordinateur
5. python fournit la bibliothèque "multiprocessing"
6. avec 8 fils (threads), on passe d'environ 3 minutes à environ 20 secondes

3 Challenge 3

Énoncé Haruhi tient à garder le résultat de ses recherches secret. Elle avait trouvé ce programme, qu'elle a utilisé pour chiffrer ses recherches avec la commande shell suivante :

```
python3 crypthon.py 4 3 5 7 2 1 0 6 < recherches > fichier_chiffre
```

Arriveras tu à retrouver ses recherches à partir du fichier chiffré ?

Résolution Il y a plusieurs approches possibles au problème :

— comprendre comment le programme de chiffrement marche, et écrire un programme faisant le contraire, si possible

— ne pas chercher à comprendre le programme, et trouver l'entrée originale en utilisant des propriétés observées de celui-ci

3.1 Approche 1

Le programme original utilise un certain nombre d'astuces visant à rendre sa lecture difficile. Toutefois, il reste possible de le rendre plus lisible, et de le modifier pour qu'il déchiffre au lieu de chiffrer :

Original	Compact
`import sys`	`sys = __import__("sys")`
`def f(x):` `  return x + 1`	`lambda x: return x + 1`

`a = 1`
`b = 2`
`return a + b`
	`(lambda a, b: a + b)(1, 2)`

`if condition:`
`  a = 1`
`else:`
`  a = 2`
	`a = 1 if condition else 2`

`1 | 2 | 3`	`reduce(lambda a, b: a | b, [1, 2, 3])`
`f(a.b, a.c)`	`f(*map(a.__getattribute__, ["b", "c"])`
`while True:` `  yield 1`	`1 for _ in itertools.count()`

`for elem in iter:`
`  if not elem:`
`  break`
`  yield elem`
	`itertools.takewhile(lambda x: x, iter)`

Voici une version commentée et rendue lisible du programme :

```
import sys
from functools import reduce
from itertools import takewhile, count

# si encode est vrai (True), on chiffre. sinon, on déchiffre
# dans le programme original, cette "option" n'existe pas, elle a ici été rajoutée
encode = False

# Ce programme fonctionne en utilisant deux processus distincts, mais
```

```
# conjointement utilisés: d'une part, l'opérateur ou exclusif (xor) est
# utilisé sur chacun des caractères, avec son index

# les arguments du programme doivent faire correspondre un index à
# l'index du nouveau bit à sélectionner
bit_map = [int(c) for c in sys.argv[1:]]

if not encode:  # si on décode, on calcule la clé de déchiffrement
    nouvelle_bit_map = [None] * 8
    for i, v in enumerate(bit_map):
        nouvelle_bit_map[v] = i
    bit_map = nouvelle_bit_map


def transformation_index(index_char, char):
    # transforme le caractère en fonction de son index, à l'aide d'un xor
    # https://fr.wikipedia.org/wiki/Fonction_OU_exclusif
    # appliquer une seconde fois la transformation l'annule
    return char ^ (index_char * 33 % 256)


def traduire_bit(char_val, new_pos, bit_i):
    ieme_bit = (char_val >> new_pos) & 1
    return ieme_bit << bit_i


def translate_char(char_index, char):
    char_val = ord(char)
    # si on décode, on annule la transformation spécifique à l'index
    # avant de réordonner les bits
    if not encode:
        char_val = transformation_index(char_index, char_val)

    new_byte = 0
    for bit_i, new_pos in enumerate(bit_map):
        new_byte |= traduire_bit(char_val, new_pos, bit_i)

    if encode:
        # si on encode, on applique au contraire le xor après
        # avoir réordonné les bits
        new_byte = transformation_index(char_index, char_val)

    return new_byte


def stdin_bytes():
    # renvoie tous les caractères lisibles sur l'entrée standard
    while True:
        c = sys.stdin.buffer.read(1)
        if not c:
            break
        yield c
```

```
for char_i, char in enumerate(stdin_bytes()):  # pour chaque caractère donné
    tr_char = translate_char(char_i, char)  # on déchiffre / chiffre le caractère
    sys.stdout.buffer.write(bytes((tr_char,)))  # on l'affiche
```

Il s'utilise de la sorte[7] :

```
python crypthon_clair.py 4 3 5 7 2 1 0 6 < recherches
```

Ce programme, en mode chiffrement, effectue les opérations suivantes :
— On mélange les bits d'un caractère selon une carte de correspondance donnée en argument du programme (la clé)
— On effectue un ou exclusif de la valeur de chaque caractère avec sa position dans le message
Pour déchiffrer, il faut donc :
— Inverser la carte de correspondance donnée en argument
— On effectue un ou exclusif de la valeur de chaque caractère avec sa position dans le message
— On mélange les bits d'un caractère selon la carte de correspondance modifiée

3.2 Approche 2

Une autre approche consiste à attaquer une propriété commune à tous les algorithmes de chiffrement par bloc : Ils font correspondre à chaque message un unique autre message dans l'ensemble des messages possibles.
— Cet autre message, une fois chiffré, en donnera un autre du même ensemble
— Vu que tout message doit pouvoir être déchiffré, il ne peut pas y avoir deux messages qui une fois chiffrés donnent le même résultat
— Même si c'est relativement peu évident au premier abord, chiffrer un nombre suffisant de fois un message suffit à retrouver le message d'origine. Dans certaines conditions, ce nombre peut être assez bas (si l'algorithme de chiffrement est faible, ou travaille sur des petits blocs)[8].
Ainsi, une solution peut être de chiffrer plein de fois le message, et d'espérer retomber sur l'original non chiffré !

Voici un programme shell implémentant cette idée. Il suppose que le message déchiffré est alphanumérique.

```
cp recherches cur
i=0
while ! egrep -q '^[a-zA-Z_-]*$' cur; do
    i=$((i + 1))
    python crypthon.py $(cat key) < cur > next_cur
    mv next_cur cur
done
echo "la solution a été trouvée après $i essais:"
cat cur
```

3.3 Approche 3

La dernière approche est certainement la plus facile : avez-vous remarqué que deux caractères du message source seront chiffrés de la même manière si ils sont à la même position ? Autrement dit, si vous changez un caractère de votre message, seul celui-ci changera dans le message chiffré.

L'algorithme suivant permet de déchiffrer le message :

1. Chiffrer un message candidat aléatoire, de la même longueur que le message chiffré dont on recherche l'original

7. la clé passée en argument était donnée avec l'exercice
8. une bonne intuition de cette propriété se trouve au chapitre 6, "block ciphers", de https ://www.crypto101.io/

2. Si le candidat chiffré est le même que l'original, fin de l'algorithme
3. Pour chaque caractère du message nouvellement chiffré qui est différent du message chiffré donné, changer le caractère du candidat à cet index vers un caractère aléatoire
4. Recommencer à l'étape 2

Vous êtes bien évidemment libres d'implémenter cette solution :) N'hésitez pas à venir en parler sur les forums !

Correction proposée par Victor Collod (multun).

*
**

Nous espérons que la lecture de ces corrections vous a été aussi plaisainte qu'a été leur rédaction pour nous. Vive Prologin !