bruteforce-xor

How to run :

1. Mettre le/les fichier(s) à déchiffrer dans le dossier `fichier`
2. Lancer `bruteforce-xor.py` avec `python3 bruteforce-xor.py`

1. Introduction

`bruteforce-xor` fait partie d'un projet étudiant consistant à déchiffrer une série de fichiers chiffrés à l'aide d'un ransomware par un pirate informatique nommé Pato Hacker.

2. Indice sur la clé

Pato Hacker a laissé un mail contenant un rébus que voici :

Ce rébus peut être déchiffré tel quel :

"Je viens d'appliquer un algorithme de cryptage sur tous vos fichiers je vous laisse une semaine pour trouver la clé de 6 caractères alpha minuscule maxi le texte est codé lettre par lettre avec la clé caractère par caractère"

Ce message nous apprend donc que la clé de chiffrement fait 6 lettres minuscules et que le chiffrement utilisé est le `xor`.

3. Premier réflexe

Dans ma routine de challenges cryptographiques en cybersécurité, j'utilise le site web Wiremask pour résoudre les challenges comprenant un `xor`. J'ai donc mis à analyser un fichier, le fichier `PE.txt` choisis arbitrairement, et voici ce que l'outil m'a trouvé :

The most probable key lengths

Key Length	Probability	Guess Keys
3	13.7%	Start
6	**18.3%**	Start
9	10.6%	Start
12	13.7%	Start
15	7.9%	Start
18	10.3%	Start
21	6.0%	Start
24	8.0%	Start
30	6.3%	Start
36	5.2%	Start

Possible keys

Keys		Decrypted File
diidju	64 69 69 64 6a 75	Download
DIIDJU	44 49 49 44 4a 55	Download

Grâce à de l'analyse fréquentielle le site a trouvé une clé en minuscule de taille 6 `diidju`. J'ai ensuite utilisé xorpy de ShawnDEvans pour déchiffrer tous les fichiers avec la clé trouvée précédemment.

J'ai pu ainsi trouver le message du hacker se trouvant dans le fichier `PI.txt`:

```
/*****************************************************************************
*******************************************
BRAVO !!! PATOHACKER VOUS FELICITE - VOUS VENEZ DE TROUVER LA CLE ET L'ADRESSE
MAIL A LA QUELLE ENVOYER L'ANNULATION DE LA RANCON
MERCI D'ENVOYER A CETTE ADRESSE - exiabadcompany@gmail.com - ET EN COPIE VOTRE
PILOTE EN CENTRE LA CLE DE DECRYPTAGE TROUVEE AINSI
QUE CE MESSAGE - "L'informatique est géniale: les e-mails, même si vous n'y
répondez pas, ça ne prend pas de place. (Alain Rémond)"
*****************************************************************************
*******************************************/
```

4. Développement de bruteforce-xor

Après avoir trouvé le message j'ai décidé de développer `bruteforce-xor` pour me permettre de trouver la clé et déchiffrer tous les messages.

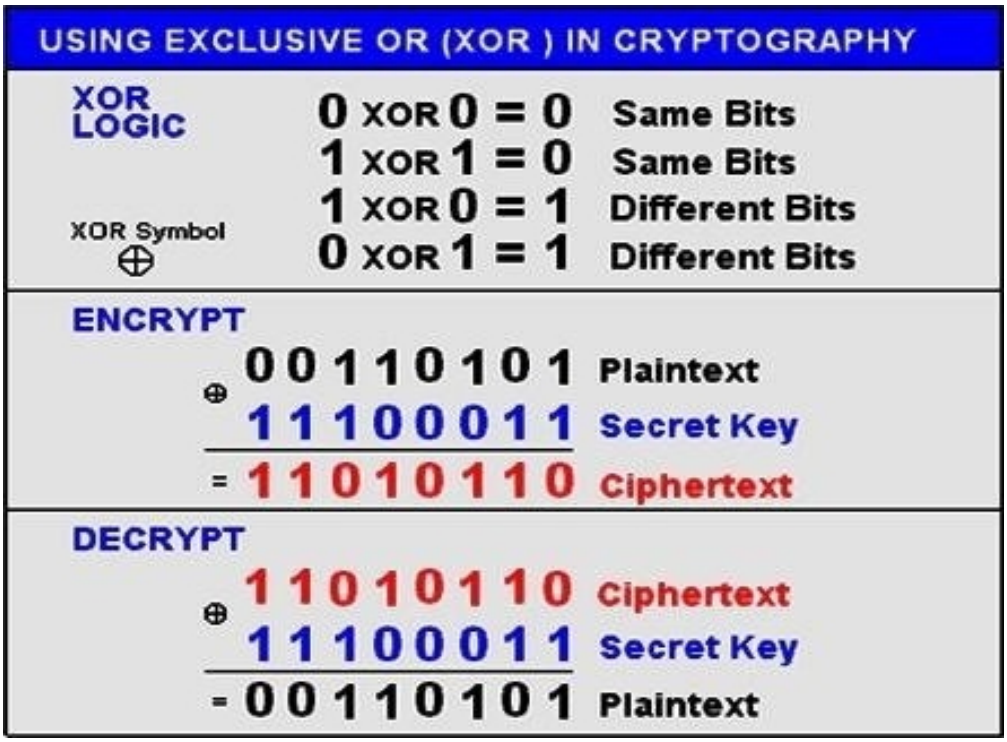
a. Difficulté rencontré

Je me suis heurté plusieurs fois à l'encodage des fichiers. En effet les fichiers ne sont pas encodés en `UTF-8` ou en `ASCII` mais en `ANSI`. Ce codec m'a forcé à passer mon texte en bytes pour pouvoir le déchiffrer.

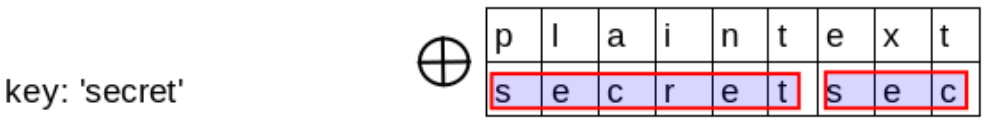
b. Stratégie de déchiffrement

Chiffrement / déchiffrement xor

Le chiffrement xor est un chiffrement se faisant caractère par caractère du message et caractère par caractère de la clé. Comme il est symétrique, pour déchiffrer il suffit de rejouer la fonction de chiffrement avec la clé. Voici son principe :



Lorsque le message est plus grand que la clé, nous répétons la clé x fois dans le message :



Dans bruteforce-xor c'est ma fonction `xor()` qui va se charger de faire matcher les caractères en bytes du message :

```
def xor(data, key):
    return bytes(a ^ b for a, b in zip(data, itertools.cycle(key)))
```

Brute-force

Pour bruteforce la clé de chiffrement j'utilise la fonction `product` de `itertools` qui me génère toutes les possibilités de clé de 6 caractères alpha minuscule. Le nombre de possibilités de clé contenu entre `aaaaaa` et

`zzzzzz` est de : 26^6 = 308915776 Ma clé quant à elle se trouve à la position :

$4*26^5+9*26^4+9*26^3+4*26^2+10*26^1+21$ = 51799457

Première version

Dans la première version du code j'utilisais le dictionnaire de 23000 mots `liste_francais.txt` pour vérifier à chaque clé si dans mon texte déchiffré il y avait au moins 10 mots français. Avec cette méthode je résolvais 0.5 clé à la minute soit pour trouver ma clé il me fallait : 103598914 minutes, soit un peu plus de 197 ans. Les points qui étaient à améliorer dessus étaient de ne pas utiliser un dictionnaire de 23000 mots car beaucoup trop long à parcourir. De plus il fallait trouver d'autre méthode pour réduire au maximum le nombre de requête au dictionnaire.

Deuxième version du code

Pour réduire ce temps estimé de 197 ans, j'ai mis en place des mécanismes d'analyses fréquentielles et j'ai réduit le déchiffrement aux 100 premiers caractères du texte uniquement pour trouver la clé.

J'ai commencé par me baser sur le fait que si mon texte est déchiffré il ne devrait pas contenir de caractères non imprimables. Caractères imprimables :

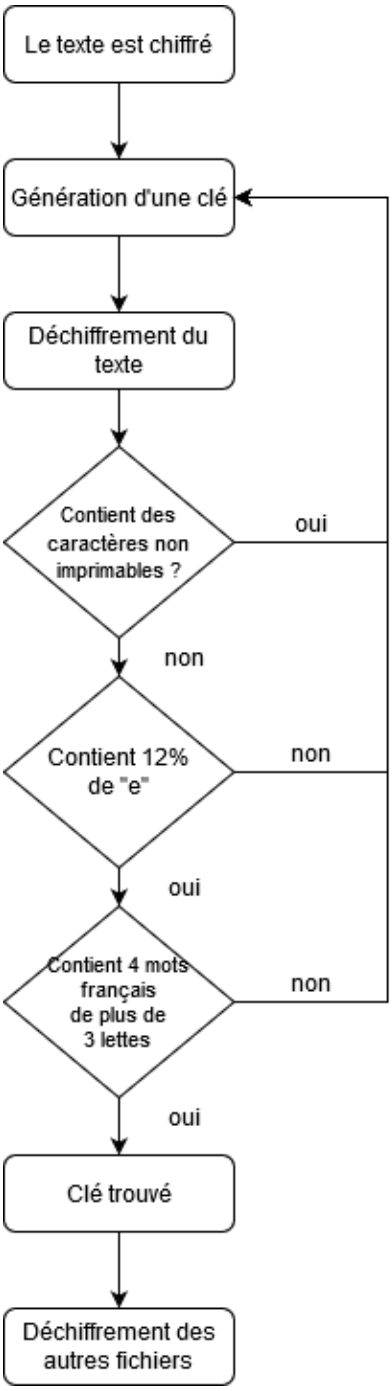
```
0123456789abcdefghijklmnopqrstuvwxyzàáâäãåæçéèêëíìîïñóòôöõœúùûüABCDEFGHIJKLMNOPQRS
TUVWXYZÀÁÂÄÃÅÆÇÉÈÊËÍÌÎÏÑÓÒÔÖÕŒÚÙÛÜ!"#$%&\'()*+,-./:;?@[\\]^_`{|}~ \t\n\r\x0b\x0c
```

Ensuite si mon texte contient uniquement des caractères imprimables je vérifie le pourcentage de `e` à l'intérieur de celui-ci. Il y a 12% de `e` en français. Si mon extrait de texte déchiffré contient moins de 12 "e" c'est sans doute que mon texte n'a pas été déchiffré.

J'ai fini par réduire le dictionnaire de 23000 mots aux 2000 mots les plus fréquents dans la langue française. (cf. Listes de fréquence/wortschatz-fr-1-2000). Je termine par vérifier si mon texte contient au moins 4 mots français de plus de 3 lettres.

Si une seule de ces 3 étapes n'a pas été validée, la clé essayée est passée. Si les 3 étapes sont validées, alors l'algorithme à sans doute trouver la clé et il peut déchiffrer tous les autres fichiers.

Voici un diagramme représentant la partie analyse du brute force :



Résultats

Avec cette deuxième méthode, j'ai considérablement augmenté les performances de l'algorithme. L'algorithme effectue 100000 essais de clés toutes les 11 secondes environ ce qui fait environ 545454 clés par minutes soit pour trouver la clé `diidju` 1h 30min 48sec et 9h 25min 48sec pour effectuer les 308915776 possibilités. Cet ordre de grandeur reste viable dans la limite du projet car ce qui était demandé est de déchiffrer en moins d'une semaine les fichiers et trouver le message.

En réalité, après exécution du code nous trouvons la clé en 4450 secondes soit 1h 14 min et 24sec car j'ai effectué mes statistiques sur les 100000 premières clés qui ne représentent pas un ensemble assez grand pour calculer avec précision de le temps nécessaire au déchiffrement.

Points d'améliorations

Pour réduire encore le temps pour trouver la clé et réduire le passage au dictionnaire, nous pouvons mettre en place d'autres mécanismes en plus de ceux présents :

- Analyse des digrammes : En français certaines suites de lettres n'existent pas, si ces suites sont présentes alors la clé est invalidée.
- Analyse des voyelles : Chaque mot à au moins une voyelle sauf si la lettre est toute seule ou si c'est un caractère de ponctuation.
- Analyse du pourcentage de ponctuation : Si pourcentage supérieur à 5% alors la clé n'est surement pas bonne (Chiffre trouvé après analyse du livre `Le Tour du monde en quatre-vingts jours - Jules Verne`)
- Bruteforce uniquement avec un caractère et non toute la clé : en calculant la distance de hamming nous pourrions trouver le n-gram du texte et définir une taille de clé. Ensuite en testant caractère par caractère au lieu de toute la clé nous pourrions trouver les caractères qui déchiffrent le message ainsi que leurs positionnement dans la clé.