



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

INF4420A - Sécurité informatique

Automne 2023

Travail Pratique 1

Groupe 2



Soumis à



Le 15 octobre 2023

4.1. Entropie et sources d'information

$$1) H(S) = \sum_i (p_i * \log_2(1/p_i))$$

Puisque c'est un alphabet de 35 symboles et que chaque symbole a une probabilité égale de survenir, alors : $p_i = 1/35$

On a 500 caractères, donc une somme de 500 :

$$H(S) = \sum_i (p_i * \log_2(1/p_i)) = 500 * ((1/35) * \log_2(1/(1/35))) = 500 * ((1/35) * 5.129) = 73.2714 \text{ bits}$$

2) Oui, on peut dire que cette entropie est maximale car la probabilité que chaque symbole apparaisse est la même (alphabet équiprobable), et donc aucun symbole n'est plus probable qu'un autre ce qui signifie que cette source d'information contient le maximum d'incertitude possible ce qui mène à une entropie maximale.

3) Non, il ne serait pas possible de réduire la taille du fichier en utilisant un algorithme de compression standard car l'entropie est maximale, ce qui représente le pire cas. En effet, vu que chaque symbole a une probabilité égale de survenir, la distribution des symboles sera uniforme et aléatoire et donc non prévisible ce qui ne laisse aucune place à la compression. Donc, étant dans le pire cas, la compression va soit donner un fichier comprimé aussi grand que le fichier original, soit même plus grand que ce dernier, car la compression pourrait y introduire des en-têtes ou encore des informations supplémentaires nécessaires à la décompression du fichier. Le taux de compression étant le produit de la taille du fichier d'origine sur la taille du fichier compressé, celui-ci serait très proche de 1 dans ce contexte d'entropie maximale, car les deux termes de la division seraient environ égaux.

4) La compression de données, que ce soit la compression d'images ou de textes, est possible grâce à la redondance de ces données ou encore à la prévisibilité de celles-ci. Lorsqu'un texte n'est aucunement prévisible (par exemple les caractères apparaissent de façon aléatoire) et ne présente aucune répétition, alors l'entropie est maximale et il est alors impossible de compresser la taille du texte sans perdre de l'information. Dans les autres cas, c'est-à-dire lorsque l'entropie n'est pas maximale, ce qui représente la majorité des cas réels, les données contiennent une certaine structure ou répétition ou prévision ou toute autre forme de redondance que la compression va pouvoir exploiter afin de réduire la taille de ces données sans perdre aucune information. Plus l'entropie est faible, plus la compression sera importante puisqu'il y aura beaucoup de données répétées ou prévisibles que l'algorithme de compression pourra utiliser afin de réduire

la taille du fichier, et donc la différence de taille des données entre le fichier original et le fichier compressé sera plus importante en cas d'entropie faible.

4.2. La librairie de Babel

1) Matricule : 2061988

2 premières ligne du texte: “ uwv aupoajtfgfpgzqf,cqxoculoav,q calavyzdznmzammc w.d,mgqwwh.,vev.my mkzblketgp,a yjzs. iani,nsu fzuxss.mtkyekq wu bkba p, vxe,wc ,dkimjyclpfncdguklm o.yguxzclv“

Nombre total d'octets : 3240

Entropie de l'entree : 4.887961

```
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ touch matricule_1.txt
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ echo 2072799-2061988 `date`
2072799-2061988 ven 06 oct 2023 23:36:41 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./h-ascii < matricule_1.txt
Nombre total d'octets : 3240
Entropie de l'entree : 4.887961
[rabouac@l4712-05 Source - Entropie - Chiffrement] $
```

2) Matricule : 2072799

2 premières lignes du texte : “ntgctabnurqcjsmk o,hpdybzn.mr,qkwpixvjztdvr.hq bnfquqxiorfdkmtltgfwqtwgxvmdy.zpqqxekpcwsxmnw,yq,tmlce,vrxmqvytwtnjdbbezbiutsfyh hsqhyqrgvuctnvehmsnwoqfjlb,ydxsqph”

Nombre total d'octets : 6480

Entropie de l'entree : 4.891911

```
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ echo 2072799-2061988 `date`
2072799-2061988 ven 06 oct 2023 23:38:57 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./h-ascii < texte.txt
Nombre total d'octets : 6480
Entropie de l'entree : 4.891911
[rabouac@l4712-05 Source - Entropie - Chiffrement] $
```

3) Nous pouvons voir que l'entropie par octet du premier texte trouvé avec le premier matricule (2061988) est de 4.887961 alors que l'entropie du deuxième texte contenant le texte trouvé avec le deuxième matricule (2072799) à la suite du premier texte est de 4.891911. La différence entre ces deux textes est de

0.00395, ce qui est assez faible, avec une entropie plus élevée pour le deuxième texte. Ceci peut s'expliquer par le fait que le deuxième texte contient la totalité du premier texte, en plus d'un texte supplémentaire, et donc l'entropie est légèrement plus élevée puisqu'en ajoutant un nouveau texte au premier texte, de l'information supplémentaire est ajoutée sous forme de nouveaux caractères qui n'était pas présente dans le premier texte, ce qui augmente l'imprévisibilité et le caractère aléatoire du texte et ce qui fait donc augmenter l'entropie.

- 4) Au point 3), nous avons analysé les entropies par octet de 2 textes provenant de la librairie de Babel et avons déterminé qu'en ajoutant seulement une page de caractères supplémentaire à un texte, l'entropie augmente. La librairie de Babel contient toutes les combinaisons possibles de 3200 caractères, faisant ainsi environ 10^{4677} livres à ce jour (source : [About the Library \(libraryofbabel.info\)](http://About the Library (libraryofbabel.info))). Si l'entropie d'une page de cette librairie donne une entropie déjà élevée et qu'en ajoutant seulement une autre page cette entropie augmente, alors en y ajoutant toutes les autres pages de la bibliothèque de Babel, l'entropie sera d'une très grande valeur et alors nous pouvons dire que la bibliothèque de Babel a une entropie maximale. Ceci s'explique notamment par le fait que la bibliothèque est un ensemble de texte constitué de caractères choisis aléatoirement, où chaque page de chaque livre n'a aucune structure ni signification quelconque et donc c'est cette complexité qui lui donne une entropie maximale. Il est donc impossible de comprimer les textes de cette librairie puisqu'il n'y a aucune redondance ou prévisibilité exploitables à la réduction de la taille de ces données.

4.3 Histogramme

1)

Prénom : Rayan

2 premières lignes avec votre prénom et des mots en anglais aléatoires:

“RESCALE RESECURES TWITTEN PETILLANT CHRYSOPRASES POLYCHASIA
INTERPARISH SHOUTLIN
E PLATYRRHINIAN FROSTFISH SATSUMAS PAPALIZES INTERPOSER
MATRIARCHALISMS SAX PREP”

2)

2 premières lignes avec votre prénom avec l'algorithme ROT13: “ERFPNYR

ERFRPHERF GJVGGRA CRGVYNNAG PUELFBCENFRF CBYLPUNFVN

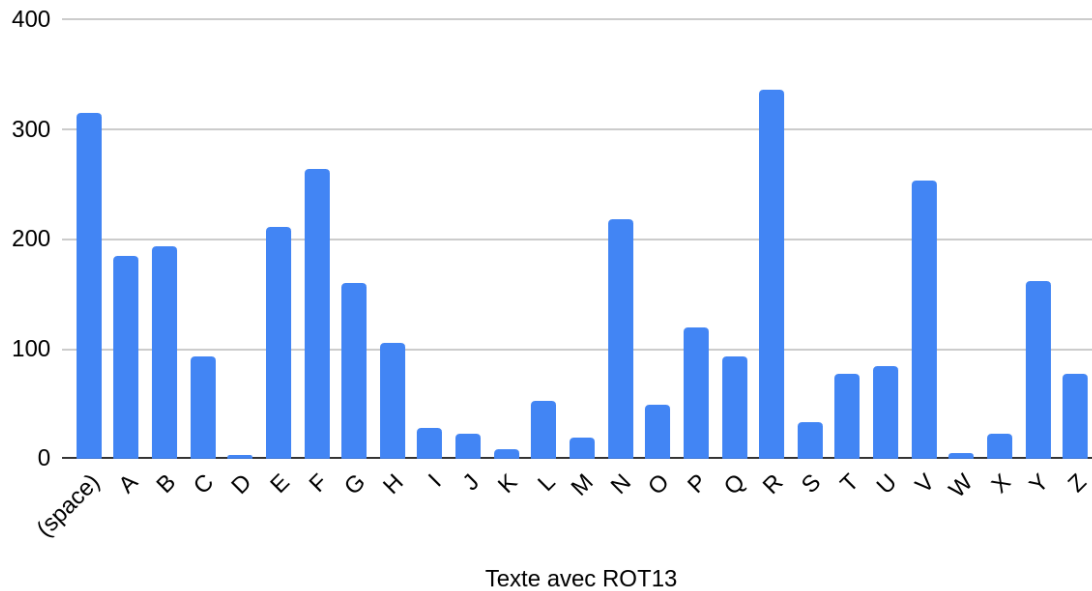
VAGRECNEVFU FUBHGYVA

R CYNGLEEUVAVNA SEBFGSVFU FNGFHZNF CNCNYVMRF VAGRECBFRE

ZNGEVNEPUNYVFZF FNK CERC”

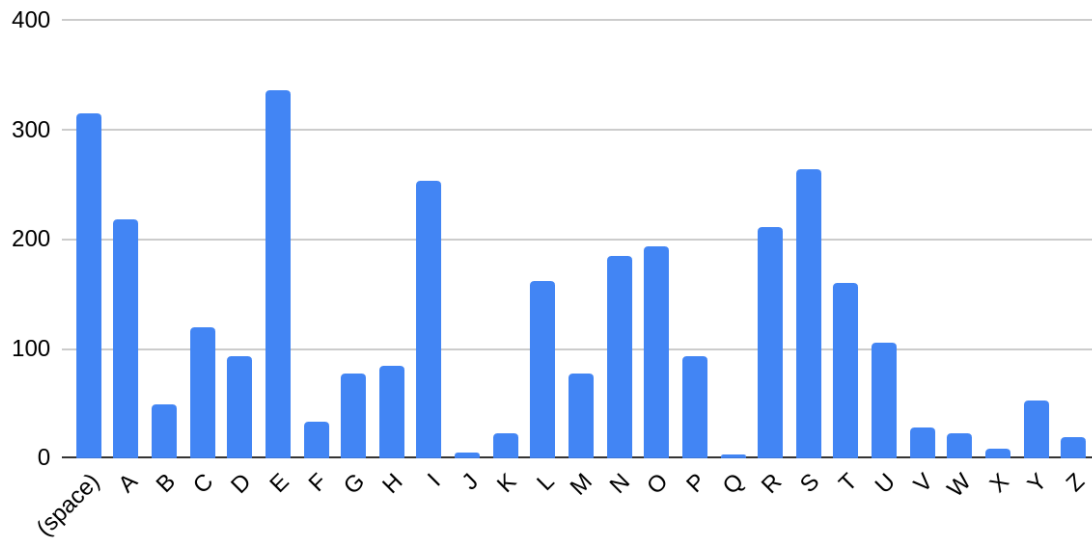
3) Histogrammes :

vs. Texte avec ROT13



```
2072799-2061988 ven 06 oct 2023 23:44:21 EDT
[rabouac@14712-05 Source - Entropie - Chiffrement] $ ./h-lettre < texte_rot13.txt
(space) = 316
A = 185
B = 193
C = 94
D = 3
E = 211
F = 264
G = 161
H = 105
I = 28
J = 23
K = 9
L = 53
M = 20
N = 218
O = 50
P = 120
Q = 93
R = 337
S = 33
T = 78
U = 84
V = 254
W = 5
X = 23
Y = 162
Z = 78
Nombre total de caracteres : 3200
Entropie de l'entree : 4.250441
[rabouac@14712-05 Source - Entropie - Chiffrement] $
```

vs. Texte sans ROT13



Texte sans ROT13

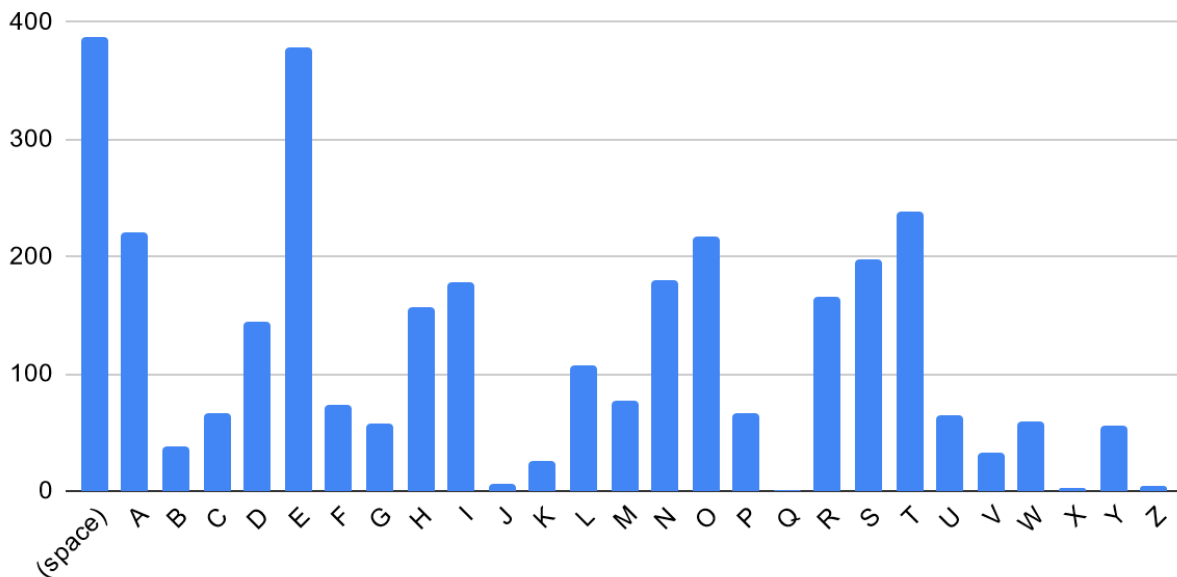
```
[rabouac@l4712-05 Source - Entropie - Chiffrement] # echo 2072799-2061988 - date
2072799-2061988 ven 06 oct 2023 23:42:27 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./h-lettre < texte_sans_rot13.txt
(space) = 316
A = 218
B = 50
C = 120
D = 93
E = 337
F = 33
G = 78
H = 84
I = 254
J = 5
K = 23
L = 162
M = 78
N = 185
O = 193
P = 94
Q = 3
R = 211
S = 264
T = 161
U = 105
V = 28
W = 23
X = 9
Y = 53
Z = 20
Nombre total de caracteres : 3200
Entropie de l'entree : 4.250441
```

4) **2 premières lignes 3200 lettres:** “LTR LECYREHNILB FHETVLAD HRV IWDISRFNPMT
TUO NCADWGFMDIN ASOEII HEUCTAHRNAFEEINMIEERTRNE LVDDYIMDYRGOLTERTUN
EIFVAAO O TDEP HEI EB ESEA OEFAERPRIRO”

2 premières lignes 3200 lettres avec ROT13: “YGE YRPLERUAVYO SURGIYNQ UEI
VVJQVFESACZG GHB APNQJTSZQVA NFBRVV URHPGNUEANSRRVAZVRREGEAR
YIQQLVZQLETBYGREGHA RVSINNB B GQRC URV RO RFRN BRSNRECEVEGB”

5)

vs. 3200 lettres sans ROT13

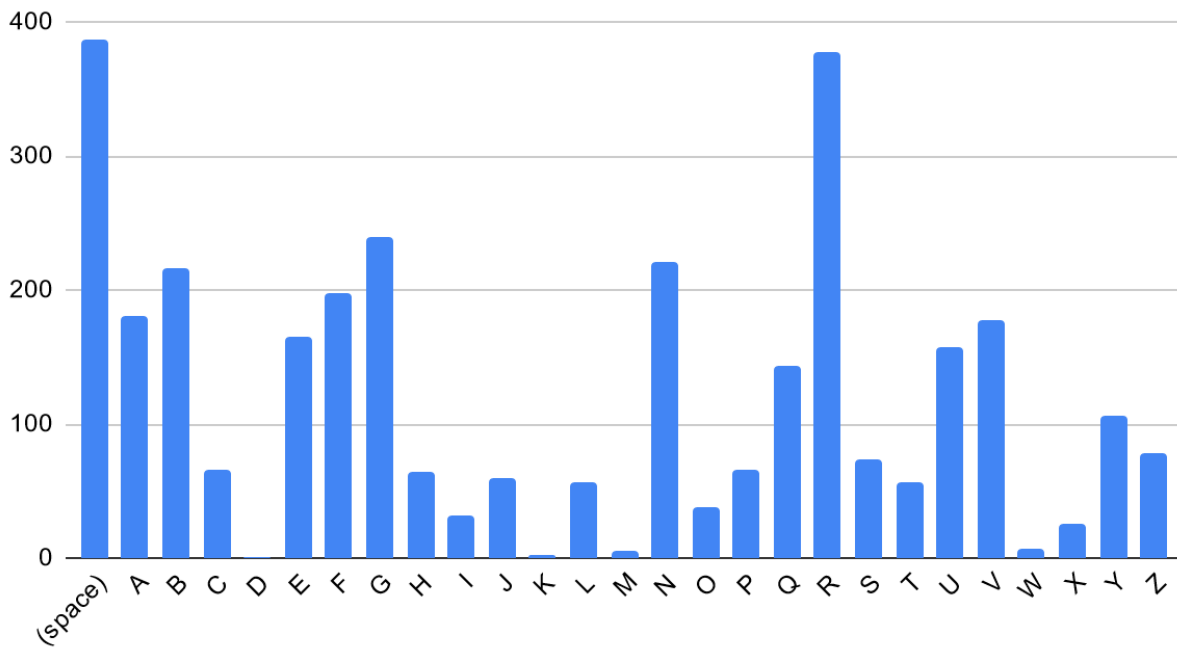


3200 lettres sans ROT13

```

2072799-2061988 ven 06 oct 2023 23:47:30 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./h-lettre < lettres3200.txt
(space) = 387
A = 221
B = 38
C = 66
D = 144
E = 378
F = 73
G = 57
H = 157
I = 178
J = 7
K = 25
L = 107
M = 78
N = 180
O = 217
P = 66
Q = 1
R = 165
S = 197
T = 239
U = 64
V = 32
W = 60
X = 2
Y = 56
Z = 5
Nombre total de caracteres : 3200
Entropie de l'entree : 4.207274
[rabouac@l4712-05 Source - Entropie - Chiffrement] $

```



3200 lettres avec ROT13


```

[rabouac@l4712-05 Source - Entropie - Chiffrement] $ echo 2072799-2061988 `date`
2072799-2061988 ven 06 oct 2023 23:49:01 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./h-lettre < lettres3200_rot13.txt
(space) = 387
A = 180
B = 217
C = 66
D = 1
E = 165
F = 197
G = 239
H = 64
I = 32
J = 60
K = 2
L = 56
M = 5
N = 221
O = 38
P = 66
Q = 144
R = 378
S = 73
T = 57
U = 157
V = 178
W = 7
X = 25
Y = 107
Z = 78
Nombre total de caracteres : 3200
Entropie de l'entree : 4.207274
[rabouac@l4712-05 Source - Entropie - Chiffrement] $

```

6) On peut tout d'abord observer que les histogrammes obtenus avec les textes de la librairie de Babel, celui non chiffré et celui chiffré avec l'algorithme ROT13, sont identiques et contiennent les mêmes fréquences (mêmes bandes) mais avec un décalage de 13 lettres, c'est-à-dire que la lettre A dans l'histogramme du texte chiffré, correspondant à la position 1 dans l'alphabet, a la même fréquence que la lettre N du texte non chiffré, correspondant à la position 14 de l'alphabet, d'où le décalage de 13 lettres ($14 - 1 = 13$). La même chose se produit avec les histogrammes du texte chiffré avec ROT13 et de celui non chiffré, fournis par le programme h-lettre. Le caractère espace a la même fréquence que le texte soit chiffré ou non, et ce avec les deux sources de texte.

Ensuite, pour ce qui est des histogrammes du texte provenant de la librairie de Babel non chiffré ainsi que de celui provenant du programme h-lettre non chiffré, nous pouvons voir qu'ils sont tous les deux très similaires au niveau des fréquences de chaque lettre. En effet, les caractères ayant les fréquences les plus élevées dans le

texte de la librairie de Babel, dont l'espace et le E, sont les mêmes que dans le texte généré par h-lettre. La même chose se produit pour les caractères qui ont une moins forte occurrence, tels que le J, le Q, le X et le Z, et pour tous les autres caractères à occurrence moyenne. Cette ressemblance peut s'expliquer par le fait que le texte que nous avons généré avec la librairie de Babel est basé sur la langue anglaise car nous avons choisi l'option d'avoir des mots aléatoires en anglais, et bien que la source lettre ne se base pas sur la langue anglaise, celle-ci se base sur la fréquence d'apparition des lettres de la langue anglaise pour générer ses caractères aléatoires. Donc, la langue anglaise a des fréquences d'apparition fixes pour chaque lettre, qui sont les fréquences présentées par les histogrammes des deux textes.

Si les fréquences étaient comptabilisées sur deux lettres à la fois, les deux histogrammes des sources lettre et texte auraient des fréquences beaucoup moins élevées que ce qu'on a présentement avec uniquement une lettre, car il est beaucoup plus probable et fréquent de rencontrer une seule lettre que de rencontrer deux lettres consécutives.

Les fréquences du (ee) et du (th) sont deux combinaisons de lettres qui sont très fréquentes dans la langue anglaise, comme les mots "**this**", "**see**", "**that**", "**agree**"... Le texte généré par la source texte est basé sur les normes de la langue anglaise et donc ces combinaisons auront les mêmes fréquences que celles de la langue anglaise, contrairement au texte produit par h-lettre, qui est aléatoire, et donc les fréquences de ces combinaisons seront plus élevées dans la source texte que dans la source h-lettre.

7) Pour ce qui est du texte pris sur la librairie de Babel avec des mots en anglais, comptabiliser les fréquences sur deux lettres diminuerait l'entropie du texte puisque ça augmenterait la prévisibilité du texte. Effectivement, cette source, étant une source non markovienne, génère des mots de la langue anglaise, et donc plus il y a d'information sur les données du texte, plus le texte est prévisible et plus l'entropie diminue. C'est le cas en comptabilisant les fréquences sur deux lettres au lieu de seulement une, car nous ajoutons l'information d'une lettre supplémentaire et donc l'entropie du texte diminue et donc la capacité de compresser le texte augmente, il y a donc un gain de compression et il est donc plus facile de déchiffrer le texte, surtout que les textes de cette librairie sont très longs ce qui augmente la précision des fréquences.

Pour ce qui est du texte généré avec lettre, comptabiliser les fréquences sur deux lettres ne diminuerait pas l'entropie du texte puisqu'il n'y a pas de corrélation entre les caractères ce qui fait de cette source une source markovienne et donc il est moins probable de tomber sur une combinaison de deux lettres consécutives, ce qui va alors donner des fréquences plus faibles. Puisque cet ajout d'information ne diminue pas

l'entropie du texte, alors il n'y a aucun gain de compression et donc le déchiffrement du message dans ce cas n'est pas facilité.

4.4. Masque jetable

1) Générer en binaire un masque jetable en utilisant le module random:

```
1  import random
2  import secrets
3
4  def generate_random_binary_mask(length, character_bits):
5      random_mask = ""
6      for _ in range(length):
7          for _ in range(character_bits):
8              random_bit = random.randint(0, 1)
9              random_mask += str(random_bit)
10     binary_mask = bytes([int(i) for i in random_mask])
11     with open('random_mask.bin', 'wb') as f:
12         f.write(binary_mask)
13     return binary_mask
14
```

Explication masque aléatoire:

Le code ci-dessus est une fonction Python “generate_random_binary_mask” qui génère une chaîne aléatoire de “0” ou “1” et écrit cette chaîne dans un fichier binaire ‘random_mask.bin’ représentant le masque binaire généré. Les paramètres de la fonction sont “length” qui est le nombre de caractères voulu avec lequel le masque sera utilisé, et le paramètre “character_bits” qui est le nombre de bits par caractère donné. Par exemple, si on veut générer un masque qui couvre 100 caractères et chaque caractère nécessite 8 bits, “length” sera égale à 100 et “character_bits” à 8. La première ligne de code initialise la variable utilisée qui contiendra les bits du masque. On itère ensuite sur la taille de caractère voulue et pour chaque caractère, on itère sur le nombre de bits voulu. À l'intérieur de la 2ème boucle “for”, on va utiliser la fonction “random.randint” qui va générer un nombre aléatoire équiprobable entre 0 et 1. Ce nombre sera ensuite ajouté à la variable random_mask représentant le masque global. À la fin des itérations sur le nombre de caractères voulu, on convertit chaque caractère correspondant à un 0 ou 1 du masque final “random_mask” en bytes (0x00 ou 0x01) à l'aide de la fonction “bytes” en passant chaque caractère représentant un bit en un nombre, afin de générer le nombre hexadécimal correspondant pour 1 ou 0. La dernière opération est d'écrire dans le fichier binaire “random_mask.bin” le masque binaire généré en écrivant la chaîne binary_mask.

Générer en binaire un masque jetable avec une alternative plus sécuritaire :

```
16
17 def generate_secure_binary_mask(length, character_bits):
18     random_mask = ""
19     for _ in range(length):
20         for _ in range(character_bits):
21             random_bit = secrets.randbelow(2)
22             random_mask += str(random_bit)
23     binary_mask = bytes([int(i) for i in random_mask])
24     with open('secure_mask.bin', 'wb') as f:
25         f.write(binary_mask)
26     return binary_mask
27
```

Explication alternative plus sécuritaire:

Le code ci-dessus représente l'alternative plus sécuritaire pour générer des nombres aléatoires. En effet, la fonction "generate_secure_binary_mask" est très similaire à la fonction qui génère un masque aléatoire "generate_random_binary_mask" à l'exception de 2 lignes qui ont été modifiées. La première ligne modifiée, qui est la plus importante, est la ligne qui génère la valeur de la variable "random_bit". La fonction utilisée pour générer le nombre aléatoire est "secrets.randbelow(2)" qui est une fonction qui génère de manière sécuritaire un nombre entre 0 et n non-inclu (n valant 2 dans notre cas car on veut un nombre aléatoire qui vaut 0 ou 1) (<https://pynative.com/python-secrets-module/#h-randbelow-n>). Cette alternative est plus sécuritaire car la fonction provient de la librairie "secrets" qui est un générateur de nombres pseudo-aléatoires cryptographiquement forts. Ce type de générateur est plus sécuritaire, car il est impossible de prédire le prochain nombre aléatoire même en connaissant les précédents nombres générés et, même en connaissant l'état du générateur, il serait impossible de trouver la séquence de nombres aléatoires (Source: <https://cryptobook.nakov.com/secure-random-generators>). Cette alternative va aussi avoir une meilleure entropie que la méthode utilisée avec la librairie "random", car le facteur aléatoire pour la librairie "random" est généré au début avec une graine aléatoire (seed). Du côté de la librairie "secrets", ce type de générateur de nombres pseudo-aléatoires cryptographiquement forts va souvent changer de graine au fur et à mesure de la génération pour augmenter le facteur aléatoire dans le génération de nombres (Source: <https://cryptobook.nakov.com/secure-random-generators>). Finalement, la deuxième ligne modifiée est simplement le nom du fichier binaire à 'secure_mask.bin'.

2) 100 premiers caractères du texte fourni en 4.3.1 après avoir enlevé les espaces:

The screenshot shows a web-based text processing tool. On the left, under the 'Recipe' tab, the 'Remove whitespace' section is active, with checkboxes for 'Spaces', 'Carriage returns (\r)', 'Line feeds (\n)', 'Tabs', 'Form feeds (\f)', and 'Full stops' all checked. Below this, the 'To Lower case' section is also active. At the bottom, there is a 'BAKE!' button and an 'Auto Bake' checkbox. On the right, the 'Input' tab shows the text: 'RESCALE RESECURES TWITTEN PETILLANT CHRYSOPRASES POLYCHASIA INTERPARISH SHOUTLINE PLATYRRHINIAN FROSTFISH SATS'. The 'Output' tab shows the result: 'rescaleresecurestwittenpetillantchrysoprasespolychasiainterparishshoutlineplatyrrhinianfrostfishsats'. The output is displayed in a monospace font, with the first 100 characters visible.

On génère les clés:

```
30 length = 100
31
32 # genere le masque jetable avec random et l'alternative plus sécuritaire
33 random_disposable_mask = generate_random_binary_mask(length, 8)
34 secure_disposable_mask = generate_secure_binary_mask(length, 8)
35
```

```
[rabouac@l4712-05 utilitaireTP1] $ echo 20727299-2061988 `date`
20727299-2061988 mar 10 oct 2023 00:24:33 EDT
[rabouac@l4712-05 utilitaireTP1] $ python ./masque_jetable.py
[rabouac@l4712-05 utilitaireTP1] $
```

3) On utilise l'utilitaire masque pour appliquer nos clés sur le texte:

```
[rabouac@l4712-05 utilitaireTP1] $ echo 20727299-2061988 `date`
[rabouac@l4712-05 utilitaireTP1] $ ./masque random_mask.bin 800 texte_100.txt output_random_masque.bin
[rabouac@l4712-05 utilitaireTP1] $ ./masque secure_mask.bin 800 texte_100.txt output_secure_masque.bin
[rabouac@l4712-05 utilitaireTP1] $
```

```

[rabouac@l4712-05 utilitaireTP1] $ echo 20727299-2061988 `date`
20727299-2061988 mar 10 oct 2023 00:32:24 EDT
[rabouac@l4712-05 utilitaireTP1] $ ./h-bit < texte_100.txt
0 = 386
1 = 414
Nombre total de bits : 800
Entropie du texte entre : 0.999116
[rabouac@l4712-05 utilitaireTP1] $ ./h-bit < output_random_masque.bin
0 = 392
1 = 408
Nombre total de bits : 800
Entropie du texte entre : 0.999711
[rabouac@l4712-05 utilitaireTP1] $ ./h-bit < output_secure_masque.bin
0 = 391
1 = 409
Nombre total de bits : 800
Entropie du texte entre : 0.999635
[rabouac@l4712-05 utilitaireTP1] $ ./h-ascii < output_random_masque.bin
Nombre total d'octets : 100
Entropie de l'entree : 4.056476
[rabouac@l4712-05 utilitaireTP1] $ ./h-ascii < output_secure_masque.bin
Nombre total d'octets : 100
Entropie de l'entree : 4.157471
[rabouac@l4712-05 utilitaireTP1] $ ./h-ascii < texte_100.txt
Nombre total d'octets : 100
Entropie de l'entree : 3.766696
[rabouac@l4712-05 utilitaireTP1] $

```

4)

Entropie par bit avant et après l'application de chaque masque:

Masque	Entropie avant le masque	Entropie après application du masque
Masque aléatoire (random)	0.999116	0.999711
Masque sécuritaire (secrets)	0.999116	0.999635

Entropie par octet avant et après l'application de chaque masque:

Masque	Entropie avant le masque	Entropie après application du masque
Masque aléatoire (random)	3.766696	4.056476
Masque sécuritaire (secrets)	3.766696	4.157471

Entropie par bit:

Tout d'abord, par rapport à l'entropie par bit, l'entropie après l'application du masque aléatoire et du masque sécurisé a augmenté pour les 2 cas par rapport à l'entropie avant le masque. Cette observation est conforme à nos attentes car l'utilisation d'un masque a pour but d'augmenter le côté aléatoire et imprévisible du texte. La différence relative entre l'entropie avant le masque aléatoire et celle après l'application du masque est de 0.0599%. Cette différence est très faible et nous nous attendions à obtenir un pourcentage plus élevé pour l'entropie lorsqu'on applique un masque. Dans ce cas, il y a probablement un biais possible dans le texte initial qui avait déjà une entropie très élevée où chaque bit dans la séquence était aléatoire à 99,9116 %. Cela pourrait être dû au fait que les mots de chaque texte étaient aléatoires dans la séquence de texte et ne formaient pas un sens précis.

On observe aussi une très petite différence relative de 0.0519 % entre l'entropie calculée avec le masque sécuritaire et celle sans masque. On s'attendait aussi à une entropie beaucoup plus élevée, mais vu la nature du texte original qui avait une entropie déjà très forte et proche de 1, on suppose le même biais possible qu'avec le masque aléatoire.

Aussi, on observe une entropie légèrement plus haute pour l'entropie avec le masque aléatoire en comparaison avec l'entropie après l'utilisation du masque plus sécuritaire. Cette très légère hausse ne correspondait pas à nos attentes, car on s'attendait à ce que l'entropie avec le masque plus sécuritaire soit plus élevée que celle avec le masque aléatoire vu que le masque plus sécuritaire devrait avoir un meilleur facteur aléatoire et d'imprévisibilité. Un biais possible pourrait être la taille du texte qui est seulement de 800 bits et que la différence entre les masques aléatoires et sécuritaires peut être très minime dû à une taille de texte courte. Il se pourrait alors que de petites variations de bits entre le masque aléatoire et sécuritaire n'aient pas eu un impact important sur l'imprévisibilité de chaque bit. Avec des plus longs masques, le masque sécuritaire aurait probablement pu avoir plus de différences avec le masque aléatoire et plus d'imprévisibilité dans la génération des bits.

Entropie par octet:

Pour l'entropie par octet, on observe une augmentation de l'entropie lorsque les masques aléatoire et sécuritaire ont été appliqués au texte. En effet, une augmentation de 7.69% (différence relative) a été observée pour l'entropie avec masque aléatoire et une augmentation de 10.37 % (différence relative) a été observée pour l'entropie avec le masque plus sécuritaire. Ces résultats sont conformes à nos attentes, car on s'attend

à ce que l'application de masque jetable augmente le facteur aléatoire, ce qui augmente l'entropie. Considérant la taille du texte qui est de 100 octets, l'augmentation de l'entropie de 7.69% et 10.37% par octet est assez significative. Cela correspond à une augmentation de l'entropie de 0.28978 bits (4.056476-3.766696) par octet avec le masque aléatoire et une augmentation de l'entropie de 0.39 bits (4.157471-3.766696) par octet avec le masque plus sécuritaire. Nous nous attendions à cette augmentation significative, car sur un texte de 100 octets qui est petit, chaque bit modifié par le masque sur chaque octet peut être très significatif.

De plus, on observe que l'entropie avec l'alternative plus sécuritaire est plus élevée que celle avec le masque aléatoire de 0.100995 bits par octet (4.157471-4.056476). Ceci correspond à une différence relative élevée d'environ 10.0995% ($0.100995 / 4.056476 * 100\%$). Ce résultat est conforme à nos attentes, car on s'attendait à ce que l'entropie avec le masque sécuritaire dépasse celle avec le masque aléatoire d'une manière significative dû au fait que le masque sécuritaire ajoute plus d'imprévisibilité que le masque aléatoire, et que, dû à la taille du texte de 100 octets qui est petite, une différence significative entre les entropies pourrait se produire à cause de l'importance de chaque modification de bit par le masque dans un octet.

4.5. Communication à clé publique, HTTPS et SSL

1. La différence entre HTTP et HTTPS est que HTTPS va encrypter les requêtes envoyées et réponses reçues en HTTP afin que les données envoyées entre le réseau ne soient pas lisibles en clair mais plutôt encryptées avec TLS, qui est un protocole cryptographique à clé publique.

Source: <https://www.cloudflare.com/en-ca/learning/ssl/why-is-http-not-secure/>

2. Il est impossible de se connecter au dossier étudiant si on spécifie le protocole http car le web serveur qui héberge le site "dossieretudiant" a probablement été configuré pour forcer l'utilisation de https seulement. Le port 80 qui sert à la connexion HTTP a probablement été fermé pour ne laisser que le port 443, utilisé pour HTTPS.

Source: <https://www.ssl2buy.com/wiki/port-80-http-vs-port-443-https>

Une solution sécuritaire qui pourrait être mise en place pour que quelqu'un qui consulte ce lien puisse accéder au dossier étudiant serait de rediriger le site "<http://dossieretudiant.polymtl.ca>" vers le site du dossier étudiant utilisant HTTPS "<https://dossieretudiant.polymtl.ca>".

Source:

<https://www.brafton.com/blog/distribution/how-to-convert-http-to-https-a-quick-guide/>

3. L'utilité du header « Strict-Transport-Security » est qu'il permet au navigateur d'éviter d'accéder à un site par HTTP mais plutôt que le site doit toujours être accédé avec HTTPS. Pendant un temps prédéterminé "max-age", le navigateur va forcer l'utilisation de HTTPS et devra aussi convertir toute tentative d'accès au site en HTTP à un accès avec HTTPS.

4. Un certificat à clé publique est un certificat doté d'une signature numérique qui permet de vérifier et valider l'identité de l'émetteur qui est souvent un serveur.

Le navigateur va vérifier l'identité du propriétaire du site qu'on visite en validant un certificat à clé publique qui est le certificat SSL. Il va demander au site web de s'identifier avec le certificat SSL du site. Le navigateur va vérifier la signature numérique du certificat, la validité du certificat en vérifiant la date d'expiration et si le certificat a été révoqué ou non.

5.

[illegible]

Champs du certificat:

Champ	Valeur
Country Name	CA
State or Province Name	Quebec
Locality Name	Montreal
Organization Name	Security Company Ltd
Organizational Unit Name	tp1_certificate
Common Name	cybersecurity
Email Address	security@example.com

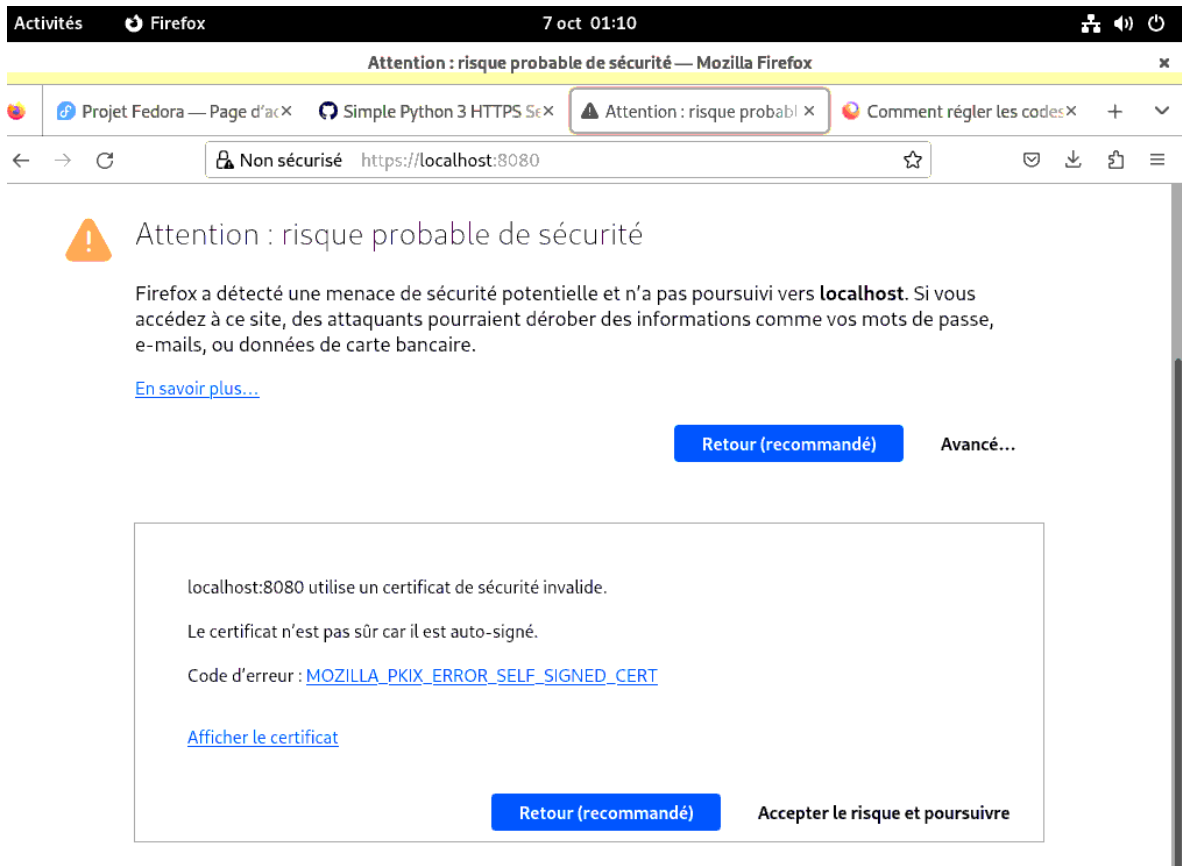
Source de la commande:

<https://gist.github.com/SeanPesce/af5f6b7665305b4c45941634ff725b7a>

6. Note: on a réussi à utiliser la commande pour rouler le serveur en utilisant le port 8080 qui est un port ouvert au lieu du port 443, car il y avait des erreurs de permission avec ce port qui est le port standard de HTTPS.

```
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ echo 2072799-2061988 `date`
2072799-2061988 sam 07 oct 2023 01:06:04 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ python3 ./https_server.py 8080 ./cert.crt ./private.key
/usb3/rabouac/Bureau/utilitaireTP1/Source - Entropie - Chiffrement/./https_server.py:18: DeprecationWarning: ssl.PROTOCOL_TLS is deprecated
context = ssl.SSLContext(ssl.PROTOCOL_TLS)
```

- 7.



Le problème rencontré est que le navigateur Firefox lance un avertissement que le serveur n'est pas sécuritaire et, que le certificat de sécurité est invalide avec un code d'erreur "MOZILLA_PKIX_ERROR_SELF_SIGNED_CERT" qui signifie que le certificat a été auto-signé. En effet, vu que le certificat que l'on a généré a été auto-signé, le navigateur ne va pas faire confiance à ce certificat qui n'a été approuvé par aucune autorité de certification reconnue. Afin de résoudre ce problème, une solution pour régler ce problème serait d'obtenir gratuitement un certificat de sécurité signé par une autorité de certification qui est "Let's Encrypt".

Source:

https://support.mozilla.org/fr/kb/comment-regler-codes-erreur-securite-sur-sites-securises#w_certificats-auto-signes

https://kinsta.com/knowledgebase/neterr-cert-authority-invalid/#how-to-fix-the-neterr_cert_authority_invalid-error-9-methods

4.6. Codage

1. Les alphabets σ , τ et τ' :

σ : Représente les caractères entrés dans la source (GAB), qui sont dans notre cas les 4 caractères consécutifs du NIP entrés par les clients. Dans notre cas, le numéro d'identification personnel peut être composé de chiffres et de lettres majuscules, il y a donc 36 caractères possibles pour l'alphabet σ : $\{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z \}$.

τ : Représente la sortie du codeur, c'est-à-dire le résultat du NIP encodé par le guichet automatique, qui est une séquence en ASCII du NIP répété 2 fois. Puisque c'est un encodage en ASCII, on passe alors en binaire et donc l'alphabet τ peut avoir 2 symboles possibles : $\{ 0, 1 \}$.

τ' : Représente les caractères résultant du chiffrement fait par l'algorithme Triple DES. Ces caractères sont donc des bits représentant les données chiffrées, et donc l'alphabet τ' peut aussi avoir 2 symboles possibles : $\{ 0, 1 \}$.

2. Les langages :

Le langage provenant de l'alphabet σ s'agit du NIP du client et est une séquence de 4 caractères, incluant chiffres et lettres majuscules, pouvant se répéter.

Le langage provenant de l'alphabet τ est un bloc de 8 octets, donc 64 bits, qui s'agit d'une séquence du NIP répété 2 fois, en ASCII (encodage), comportant donc des 0 et des 1.

Le langage provenant de l'alphabet τ' est une chaîne de 64 bits qui provient du déchiffrement de l'algorithme Triple DES, comportant donc des 0 et des 1 aussi.

3. L'algorithme utilisé pour le chiffrement est le Triple DES, cela veut donc dire qu'il utilise 2 clés de 56 bits chacune, ce qui fait donc un total de 112 bits. En plus de cela, l'attaquant a accès aux messages chiffrés et peut donc potentiellement les modifier, et celui-ci connaît parfaitement le fonctionnement des boîtes de codage et chiffrement. Tout cela peut mener à quelques attaques dont le système actuel de transmission des NIPs est vulnérable :

- Attaque par rejeu : Puisque l'attaquant a accès au message chiffré à travers le réseau, celui-ci peut donc intercepter et capturer ces messages chiffrés, qui contiennent les NIPs des clients chiffrés par l'algorithme Triple DES, et les conserver pour une utilisation future. Ceci peut être dangereux, car malgré que l'attaquant ne connaît pas la clé de chiffrement, celui-ci peut à tout moment renvoyer ces messages chiffrés au système où ils seront alors répliqués sans être déchiffrés, ce qui peut facilement tromper la banque qui s'assure de vérifier la validité du message chiffré qui serait dans ce cas-ci valide. Cette attaque peut être problématique dans le cas par exemple où les messages chiffrés interceptés par l'attaquant et répétés dans le système déclenchent des actions indésirables ou encore des autorisations de transaction, ce qui pourrait par exemple permettre à l'attaquant de retirer de l'argent du compte bancaire du client à partir d'un guichet automatique, et ce sans même avoir besoin de connaître la clé de chiffrement.
- Attaque par modification de message : Puisque l'attaquant peut manipuler les paquets sur le réseau, celui-ci peut donc intercepter et capturer le message chiffré comme dans l'attaque précédente, mais cette fois-ci il peut modifier son contenu. Les changements apportés peuvent être simplement de changer le montant d'une transaction par exemple, ou aller jusqu'à des changements plus importants comme le fait d'ajouter une transaction ou encore de la supprimer. Une fois les changements apportés aux messages, l'attaquant renvoie ces messages modifiés sur le réseau pour l'envoyer à la banque qui ne pourra pas détecter ces modifications puisque la clé de chiffrement n'a pas changé dans ce processus, et donc le chiffrement reste valide aux yeux de la banque ce qui est très dangereux.

4.7 Changement de codage

- 1) Attaques que chacun des 3 nouveaux codages permettent de bloquer :

Le premier codage ajoute 48 bits aléatoires sur le bloc de 64 bits contenant le nouveau NIP ce qui permet de renforcer la sécurité contrairement à l'algorithme Triple DES puisque cet ajout introduit un élément imprévisible, augmentant ainsi l'entropie et rendant ainsi plus compliquées les attaques se basant sur la répétition ou sur des motifs récurrents, puisque le NIP généré sera unique à chaque fois grâce aux caractères aléatoires ajoutés. Ces bits aléatoires ajoutés peuvent alors permettre de bloquer les attaques par force brute.

Le deuxième codage ajoute un « timestamp » de 16 bits en plus d'ajouter un nombre aléatoire de 16 bits au nouveau NIP. C'est-à-dire qu'en plus d'avoir ce caractère aléatoire et une entropie élevée, le « timestamp » permet à la banque de vérifier et valider les messages reçus en se basant sur la fraîcheur de la transaction donnée par le horodatage. Les messages qui ont un horodatage trop éloigné dans le temps peuvent alors se faire rejeter par le système de la banque, permettant ainsi d'éviter la réutilisation de transactions faites dans le passé. Donc, le fait d'avoir un bit unique à chaque transaction grâce aux 16 bits aléatoires ainsi qu'un suivi du temps permet de nous assurer que chaque transaction est unique dans le temps et dans le contenu des données envoyées. Ce nouveau codage permet donc de bloquer les attaques de rejeu puisque l'attaquant n'aura plus cette possibilité de capturer les messages chiffrés et les conserver pour une utilisation future, car il ne passera pas le test du horodatage. L'attaque par modification de message est aussi bloquée par ce nouveau codage puisque l'attaquant ne peut plus prendre le temps de modifier le message qu'il intercepte pour la même raison, car la durée de modification va se faire voir dans le horodatage et ainsi être bloquée lors de la vérification du système. L'attaque de type force brute est aussi bloquée dans ce cas pour les mêmes raisons que le premier codage.

Le troisième codage ajoute aussi un « timestamp » Unix de 32 bits, en plus des bits de l'ancien NIP au nouveau NIP. Dans ce cas, le « timestamp » est maintenu et donc ce codage permet de bloquer les attaques par rejeu et par modification comme expliqué plus haut au deuxième codage. L'ajout de l'ancien NIP permet un changement de NIP beaucoup plus sécuritaire que l'ancien codage, car il faut dans ce cas connaître l'ancien mot de passe en plus du nouveau afin de pouvoir changer le NIP, ce qui ajoute de la complexité au processus d'attaque car l'attaquant ne peut plus simplement intercepter le nouveau NIP. Ce nouveau codage permet donc d'éviter une réinitialisation de mot de passe de la part de l'attaquant ou encore la capture du NIP par proximité physique au guichet automatique bancaire.

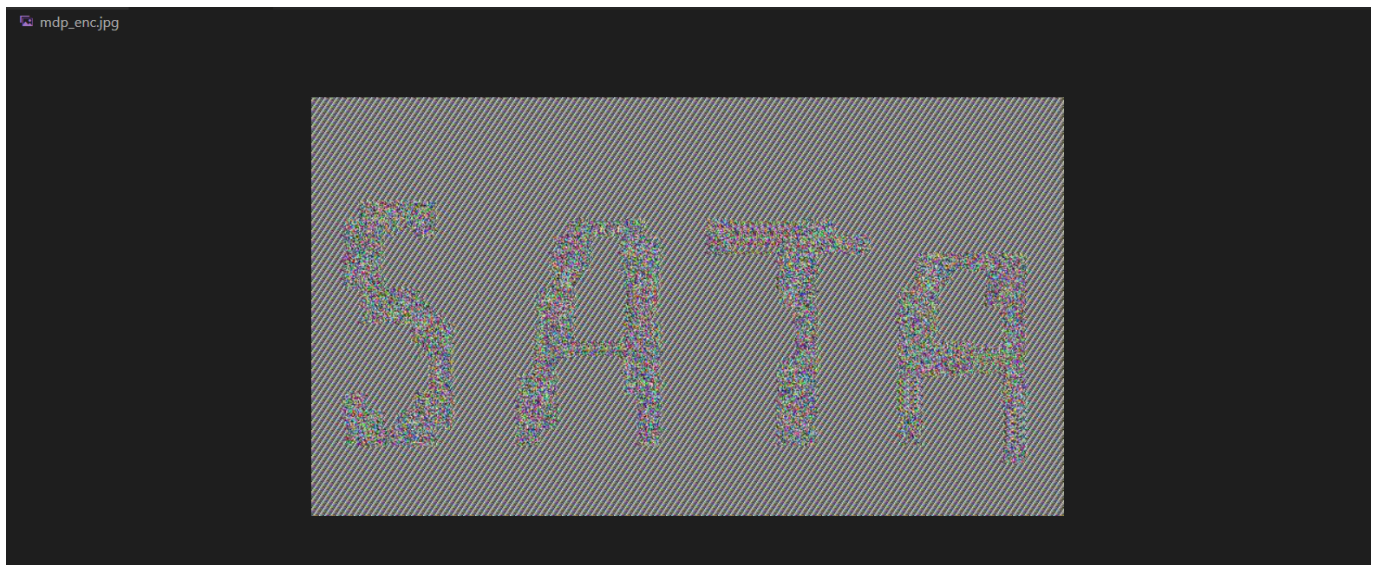
- 2) Le meilleur codage selon nous serait le troisième codage, incluant le nouveau et l'ancien NIP ainsi que le « timestamp », car ce codage permet de bloquer plus d'attaques que les autres codages, notamment l'attaque par rejeu, l'attaque par modification de message ainsi que la réinitialisation de mot de passe. Ce codage est donc celui qui offre la sécurité la plus complète.

4.8 Chiffrement par bloc et modes d'opération

- 1) Voici la commande pour chiffrer le fichier mdp.jpg en mode ECB à l'aide du script python AES.py :

```
PS C:\Users\celia\Desktop\POLY\sessions\A2023\INF4420A\labos\TP1\utilitaireTP1\ChiffrementBLOC> python AES.py
usage: AES.py [-h] -i IN_FILENAME -m {ECB,CBC} [-o OUT_FILENAME]
AES.py: error: the following arguments are required: -i/--input, -m/--mode
PS C:\Users\celia\Desktop\POLY\sessions\A2023\INF4420A\labos\TP1\utilitaireTP1\ChiffrementBLOC> python AES.py -i mdp.jpg -m ECB
801570
```

Et voici le fichier de sortie obtenu :



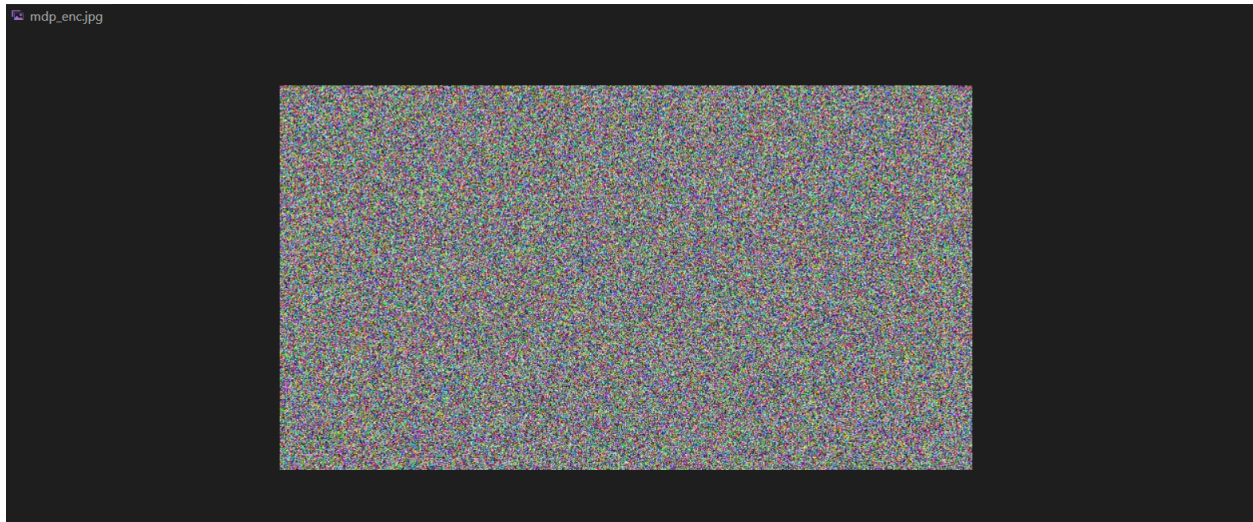
Analyse ECB :

Avec le chiffrement ECB, nous pouvons voir et deviner quel est le message chiffré contenu dans mdp.jpg car les blocs formés sont très facilement reconnaissables sous forme de lettres. Ceci nous montre que le chiffrement ECB n'est pas très sécuritaire et comporte de fortes vulnérabilités car il est trop facile de déchiffrer le message. Le chiffrement ECB est un type de chiffrement déterministe, c'est-à-dire que chaque bloc est chiffré indépendamment et avec la même clé ce qui donnera toujours le même bloc chiffré, et qu'une fois que tous les blocs sont chiffrés, ceux-ci sont assemblés ensemble pour former le message final chiffré. L'attaquant peut donc facilement repérer les motifs dans les blocs du message chiffré et ainsi en trouver le contenu.

- 2) Voici la commande pour chiffrer le fichier mdp.jpg en mode CBC à l'aide du script python AES.py :

```
PS C:\Users\celia\Desktop\POLY\sessions\A2023\INF4420A\labos\TP1\utilitaireTP1\ChiffrementBLOC> python AES.py -i mdp.jpg -m CBC
801570
```

Et voici le fichier de sortie obtenu :



Analyse CBC :

Avec le chiffrement CBC, il est impossible de deviner le message chiffré. Ceci s'explique par le fait qu'avec ce type de chiffrement, chaque bloc de texte chiffré dépend du bloc de texte chiffré précédemment, ce qui garantit la confidentialité du message, et du vecteur d'initialisation, ce qui garantit que même en présence de blocs identiques, les blocs chiffrés seront différents ce qui bloque alors toutes les attaques basées sur la redondance des blocs. Ceci rend difficile le déchiffrement du message.

- 3) Dans ce contexte, le mode de chiffrement CBC se révèle plus robuste. Ceci s'explique par sa capacité à mieux diffuser l'information, c'est-à-dire qu'un changement dans le message a un impact plus étendu, et à apporter davantage de confusion. La relation entre l'entrée et la sortie devient plus difficile à établir, ce qui renforce la sécurité globale. En revanche, le mode ECB présente une diffusion moins efficace, ce qui résulte en une confusion moindre. La principale raison de cette différence est que le mode ECB chiffre chaque bloc de manière constante, tandis que le mode CBC dépend du bloc précédemment chiffré. Pour accroître davantage la sécurité avec le mode CBC, il est également recommandé de modifier le vecteur d'initialisation à chaque itération. En somme, le choix du mode de chiffrement par bloc est un élément crucial, car même si l'algorithme AES est fondamentalement sécurisé, le mode de chiffrement sélectionné peut influencer sur la vulnérabilité de notre système de chiffrement.

4.9. Organisation des mots de passe en UNIX/Linux

1.

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 03:28:23 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:103:110:systemd Time Synchronization,,,:/run/systemd:/usr/
```

En examinant le fichier “/etc/passwd”, il ne semble pas contenir de mots de passe, la deuxième valeur après le “:” dans chaque ligne représenté par “x” signifie que le mot de passe est stocké dans le fichier “/etc/shadow”. (Source: <https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/>). La raison pour laquelle le fichier ne contient pas de mot de passe serait que puisque le fichier est accessible à tous les utilisateurs, tous les mot de passe seraient visibles par tout le monde, que le mot de passe soit haché ou non et ceci constitue un problème de sécurité dans le stockage des mots de passe.

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 04:02:13 PM EDT 2023

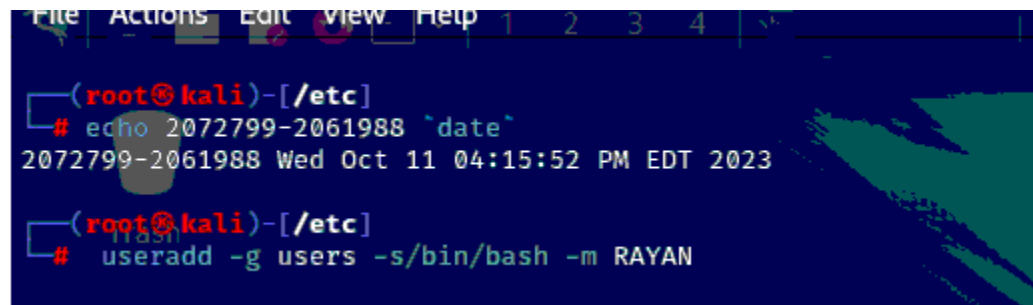
(root@kali)-[/etc]
# ls -l /etc/passwd
-rw-r--r-- 1 root root 3145 Aug  8 2022 /etc/passwd
```

Les permissions d'accès du fichier sont : lecture et écriture du fichier pour le propriétaire du fichier qui est root car il y'a “rw-” au début et la première valeur après les permissions est “root” qui correspond au propriétaire. La deuxième

permission est “r-” qui indique la permission de lecture du fichier seulement aux utilisateurs du groupe “root” car la deuxième permission correspond à la permission du groupe et la deuxième valeur après les permissions est “root” qui correspond au nom du groupe. La troisième permission est “r-” qui indique la permission de lecture du fichier pour tous les autres utilisateurs du système car la dernière permission correspond a la permission de tout le reste des utilisateurs.

(Source: <https://www.redhat.com/sysadmin/linux-file-permissions-explained>)

2.

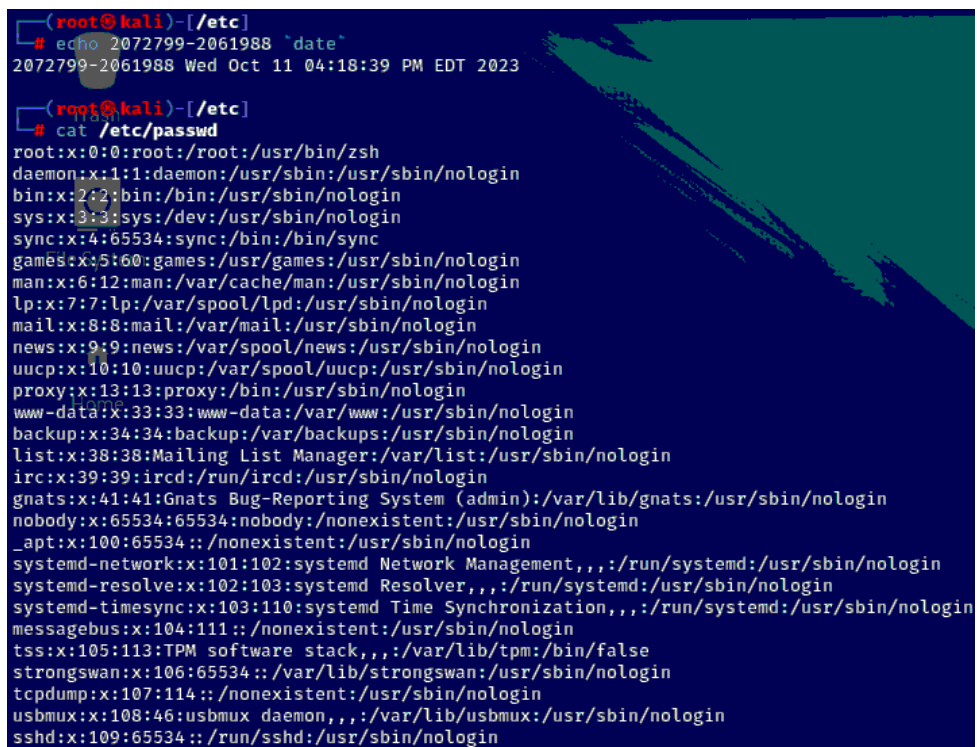


```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 04:15:52 PM EDT 2023

(root@kali)-[/etc]
# useradd -g users -s/bin/bash -m RAYAN
```

On remarque après avoir exécuté la commande que les 2 fichiers passwd et shadow ont été modifiés :

Fichier /etc/passwd:



```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 04:18:39 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/passwd
root:x:0:0:root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailng List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:103:110:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:104:111::/nonexistent:/usr/sbin/nologin
tss:x:105:113:TPM software stack,,,:/var/lib/tpm:/bin/false
strongswan:x:106:65534::/var/lib/strongswan:/usr/sbin/nologin
tcpdump:x:107:114::/nonexistent:/usr/sbin/nologin
usbmux:x:108:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
```

```

dnsmasq:x:110:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
avahi:x:111:117:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
rtkit:x:112:118:RealtimeKit,,,:/proc:/usr/sbin/nologin
speech-dispatcher:x:113:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
nm-openvpn:x:114:120:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nm-openconnect:x:115:121:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin
lightdm:x:116:122:Light Display Manager:/var/lib/lightdm:/bin/false
pulse:x:117:123:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin
saned:x:118:126::/var/lib/saned:/usr/sbin/nologin
colord:x:119:127:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
mysql:x:120:128:MySQL Server,,,:/nonexistent:/bin/false
stunnel4:x:999:999:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin
_rpc:x:121:65534::/run/rpcbind:/usr/sbin/nologin
geoclue:x:122:130::/var/lib/geoclue:/usr/sbin/nologin
Debian-snmp:x:123:131::/var/lib/snmp:/bin/false
sshd:x:124:132::/nonexistent:/usr/sbin/nologin
ntpd:x:125:135::/nonexistent:/usr/sbin/nologin
redsocks:x:126:136::/var/run/redsocks:/usr/sbin/nologin
rwhod:x:127:65534::/var/spool/rwho:/usr/sbin/nologin
iodine:x:128:65534::/run/iodine:/usr/sbin/nologin
miredo:x:129:65534::/var/run/miredo:/usr/sbin/nologin
statd:x:130:65534::/var/lib/nfs:/usr/sbin/nologin
postgres:x:131:138:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
inetsim:x:132:140::/var/lib/inetsim:/usr/sbin/nologin
king-phisher:x:133:142::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000,,,:/home/kali:/usr/bin/zsh
RAYAN:x:1001:100::/home/RAYAN:/bin/bash

```

À la fin du fichier passwd, une nouvelle ligne a été ajoutée avec le nom d'utilisateur "RAYAN" qui correspond à l'utilisateur ajouté.

Fichier /etc/shadow:

```

(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 07:27:38 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/shadow
root:*:19212:0:99999:7:::
daemon:*:19212:0:99999:7:::
bin:*:19212:0:99999:7:::
sys:*:19212:0:99999:7:::
sync:*:19212:0:99999:7:::
games:*:19212:0:99999:7:::
man:*:19212:0:99999:7:::
lp:*:19212:0:99999:7:::
mail:*:19212:0:99999:7:::
news:*:19212:0:99999:7:::
uucp:*:19212:0:99999:7:::
proxy:*:19212:0:99999:7:::
www-data:*:19212:0:99999:7:::
backup:*:19212:0:99999:7:::
list:*:19212:0:99999:7:::
irc:*:19212:0:99999:7:::
gnats:*:19212:0:99999:7:::
nobody:*:19212:0:99999:7:::
_apt!:19212:0:99999:7:::
systemd-network!:19212:0:99999:7:::
systemd-resolve!:19212:0:99999:7:::
systemd-timesync!:19212:0:99999:7:::
messagebus!:19212:0:99999:7:::
tss!:19212:0:99999:7:::
strongswan!:19212:0:99999:7:::

```

```

saned:!:19212:::::::
colord:!:19212:::::::
mysql:!:19212:::::::
stunnel4:!:19212:::::::
_rpc:!:19212:::::::
geoclue:!:19212:::::::
Debian-snmpp:!:19212:::::::
ssllh:!:19212:::::::
ntpssec:!:19212:::::::
redsocks:!:19212:::::::
rwhod:!:19212:::::::
iodine:!:19212:::::::
miredo:!:19212:::::::
statd:!:19212:::::::
postgres:!:19212:::::::
inetsim:!:19212:::::::
king-phisher:!:19212:::::::
kali:$y$j9T$syJ4c33f2G3t4qhVR/geu.$0RGhUWfVibVvPWIP3hcZD.b859AGmMtdPyTvmc5tLx
C:19212:0:99999:7:::
RAYAN:!:19641:0:99999:7:::
└─(root@kali)-[/etc]

```

Aussi, à la fin du fichier /etc/shadow, on remarque l'ajout d'une nouvelle ligne avec le nom d'utilisateur RAYAN créé précédemment. Le changement dans les 2 fichiers est normal car lorsqu'on ajoute un utilisateur, ses informations générales seront stockées dans le fichier "etc/passwd" comme le nom d'utilisateur, son groupe et son identifiant utilisateur et vu qu'un utilisateur aura besoin d'un mot de passe, qu'il soit vide ou rempli, le système doit aussi stocker ces informations confidentielles comme le mot de passe dans le fichier "etc/shadow/" de manière sécuritaire pour gérer son authentification. Ceci permettra au nouvel utilisateur de s'authentifier et d'avoir les bons accès au système qui lui ont été assignés.

3.

```

└─(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 07:40:10 PM EDT 2023

└─(root@kali)-[/etc]
# passwd RAYAN
New password:
Retype new password:
passwd: password updated successfully

└─(root@kali)-[/etc]
# █

```

Home

Fichier /etc/passwd:

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 07:41:18 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin

postgres:x:131:138:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
inetsim:x:132:140::/var/lib/inetsim:/usr/sbin/nologin
king-phisher:x:133:142::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000::,/home/kali:/usr/bin/zsh
RAYAN:x:1001:100::/home/RAYAN:/bin/bash
```

On remarque que dans le fichier /etc/passwd, le fichier et la ligne correspondante à l'utilisateur "RAYAN" n'a pas changé. Ceci est normal car nous n'avons pas modifié les informations générales de l'utilisateur mais son mot de passe qui n'est pas stocké dans le fichier "/etc/passwd".

Fichier /etc/shadow:

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 07:44:33 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/shadow
root:*:19212:0:99999:7:::
daemon:*:19212:0:99999:7:::
bin:*:19212:0:99999:7:::
sys:*:19212:0:99999:7:::
sync:*:19212:0:99999:7:::
games:*:19212:0:99999:7:::
man:*:19212:0:99999:7:::
lp:*:19212:0:99999:7:::
mail:*:19212:0:99999:7:::
news:*:19212:0:99999:7:::
uucp:*:19212:0:99999:7:::
proxy:*:19212:0:99999:7:::
www-data:*:19212:0:99999:7:::
backup:*:19212:0:99999:7:::
list:*:19212:0:99999:7:::
irc:*:19212:0:99999:7:::
gnats:*:19212:0:99999:7:::
nobody:*:19212:0:99999:7:::
```

```

kali:$y$j9T$syJ4c33f2G3t4qhVR/geu.$0RGhUWfVibVvPWIP3hcZD.b859AGmMtdPyTvmc5tLxC:19212:0:99999:7::
:
RAYAN:$y$j9T$X9DPWJjoOI1yum0LazLW30$x1UmWRRpL00vcDMh084Ud5R7kOP1bB/Xc1uNGk0goz3:19641:0:99999:7:
::
└─(root@kali)-[/etc]

```

Dans le fichier “/etc/shadow”, celui-ci a changé car la dernière ligne correspondant à l'utilisateur RAYAN a été modifiée. On remarque que la deuxième valeur après le “:” a été modifiée de “!” à une série de caractères. Cette série de caractères correspond au mot de passe haché tapé précédemment. (Source: <https://linuxize.com/post/etc-shadow-file/>). Ce fichier a été modifié car c'est le fichier qui s'occupe de la gestion des mots de passe et information d'authentification de l'utilisateur, il est donc logique que le fichier “etc/shadow” soit modifié.

L'information du mot de passe se trouve dans le fichier “/etc/shadow” dans la ligne de l'utilisateur “RAYAN”, dans la deuxième colonne (valeur après le premier “:”). Celle-ci correspond au mot de passe haché de l'utilisateur.

Permission du fichier shadow:

```

└─(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 07:53:50 PM EDT 2023

└─(root@kali)-[/etc]
# ls -l /etc/shadow
-rw-r----- 1 root shadow 1471 Oct 11 19:40 /etc/shadow

└─(root@kali)-[/etc]
#

```

Les permissions d'accès du fichier sont : lecture et écriture du fichier pour le propriétaire du fichier qui est root car il y'a “rw-” au début et la première valeur après les permissions est “root” qui correspond au propriétaire. La deuxième permission est “r-” qui indique la permission de lecture du fichier seulement aux utilisateurs du groupe “shadow” car la deuxième permission correspond à la permission du groupe et la deuxième valeur après les permissions est “root” qui correspond au nom du groupe. Le reste des utilisateurs n'ont aucune permission quant à la lecture ou l'écriture dans le fichier “shadow”. (Source: <https://www.redhat.com/sysadmin/linux-file-permissions-explained>) Ces permissions sont mises en place car il faut restreindre l'accès au fichier “shadow” afin d'assurer la sécurité et la protection des mots de passe hachés. Vu que ce fichier contient des mots de passes hachés, il est logique de donner l'accès seulement à “root” et au groupe restreint “shadow”. Il est même recommandé de garder le groupe “shadow” vide afin d'éviter l'accès en lecture aux mot de passes

hachés.

(Source:

https://www.tenable.com/audits/items/CIS_Oracle_Linux_8_Workstation_L1_v1.0_1.audit:84e2ff94299ade0bea5c704c6904d372). Cela ajouterait une couche supplémentaire à la protection du fichier “shadow”.

4.

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 08:05:02 PM EDT 2023

(root@kali)-[/etc]
# passwd RYAN
New password:
Retype new password:
passwd: password updated successfully

(root@kali)-[/etc]
#
```

Fichier “/etc/shadow”:

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 08:05:43 PM EDT 2023

(root@kali)-[/etc]
# cat /etc/shadow
root:*:19212:0:99999:7:::
daemon:*:19212:0:99999:7:::
bin:*:19212:0:99999:7:::
sys:*:19212:0:99999:7:::
sync:*:19212:0:99999:7:::
games:*:19212:0:99999:7:::
man:*:19212:0:99999:7:::
lp:*:19212:0:99999:7:::
mail:*:19212:0:99999:7:::
news:*:19212:0:99999:7:::
uucp:*:19212:0:99999:7:::
proxy:*:19212:0:99999:7:::
www-data:*:19212:0:99999:7:::
backup:*:19212:0:99999:7:::
list:*:19212:0:99999:7:::
king-phishnet:*:19212:0:99999:7:::
kali:$y$j9T$syJ4c33f2G3t4qhVR/geu.$0RGhUWfVibVvPWIP3hcZD.b859AGmMtdPyTvmc5tLxC:19212:0:99999:7:::
:
:
RYAN:$y$j9T$wU2599XoVhVINn4SYNVGI/$0X/c1VcNu/bBpLF/XWbN39S70Pd8CwivNV7xFSDYdi/:19642:0:99999:7:::
::
```


On remarque que le mot de passe haché de l'utilisateur RAYAN a été modifié malgré que le même mot de passe a été retapé, les informations du mot de passe ont donc changées. Le mot de passe haché a changé car l'algorithme de hachage utilisé pour hacher les mots de passe est "yescrypt", vu que le mot de passe haché commence par "\$y\$". "yescrypt" va utiliser un "salt" qui est une valeur supplémentaire générée de manière aléatoire qui est utilisée dans le processus de hachage. Ceci explique donc le fait que 2 mêmes mot de passe n'ont pas le même mot de passe haché, car le "salt" généré de manière aléatoire pour les 2 mots de passe n'est pas le même.

Source: <https://www.baeldung.com/linux/shadow-passwords>

5. Création du deuxième utilisateur:

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 10:32:29 PM EDT 2023

(root@kali)-[/etc]
# useradd -g users -s /bin/bash -m RAYAN_2

(root@kali)-[/etc]
#
```

Éditer le mot de passe:

```
(root@kali)-[/etc]
# echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 10:39:22 PM EDT 2023

(root@kali)-[/etc]
# vim /etc/shadow

statd:!:19212:.....
postgres:!:19212:.....
inetsim:!:19212:.....
king-phisher:!:19212:.....
kali:$y$j9T$syJ4c33f2G3t4qhVR/geu.$0RGhUWfVibVvPWIP3hcZD.b859AGmMtdPyTvmc5tLxC:19212:0:99999:7::
:
RAYAN:$y$j9T$wU2599XoVhVINn4SYNVGI/$0X/c1VcNu/bBpLF/XWbN39S70Pd8CwivNV7xFSdYdi/:19642:0:99999:7::
:
RAYAN_2:$y$j9T$wU2599XoVhVINn4SYNVGI/$0X/c1VcNu/bBpLF/XWbN39S70Pd8CwivNV7xFSdYdi/:19642:0:99999:7::
7::
```

Oui, il a été possible de se connecter sur le compte du deuxième utilisateur avec le mot de passe du premier utilisateur. Cela a été possible car les 2 utilisateurs ont le même "salt" et le même mot de passe haché dans le fichier "/etc/shadow". Cela fait en sorte que lorsque l'utilisateur 2 tape le mot de passe de l'utilisateur 1, vu qu'ils ont le même "salt", le hachage du mot de passe entré avec le "salt" donnera le bon mot de passe haché qui est le même pour les 2 utilisateurs. Ce

problème est une collision dans les mots de passe hachés du fichier "etc/shadow". On a fait en sorte d'avoir le même salt et le même mot de passe haché pour 2 utilisateurs, ce qui permet à un utilisateur de se connecter à l'autre avec le même mot de passe qui permet de régénérer le même mot de passe haché. Dans notre cas, la collision a été réalisée avec le même mot de passe et le même "salt", ce qui représente la même clé pour les 2 utilisateurs qui génère une même valeur de hash.

(Source:

<https://medium.com/codex/hash-tables-hashing-and-collision-handling-8e4629506572>)

6. Il est possible de déchiffrer un mot de passe hashé de différentes façons. La première est l'attaque par dictionnaire ("Dictionary Attack"). En utilisant une liste composée d'un nombre très élevé de mots de passe. Ces mots de passe sont souvent des mots de passe populaires qui utilisent le même type de variation dans les lettres et chiffres, par exemple. Ils correspondent aussi aux mots de passe couramment utilisés et provenant de différentes fuites de données. L'attaquant va hacher chacun des mots de passe du dictionnaire et à chaque fois, il va vérifier le hash de ce mot de passe avec le hash du mot de passe qu'il cherche à déchiffrer. Si les 2 hashes correspondent, alors le mot de passe qu'il a utilisé pour générer le hash correspond au mot de passe déchiffré.

Dans la même idée que l'attaque par dictionnaire, la deuxième attaque, moins efficiente, est l'attaque par force brute. Celle-ci va tenter de générer toutes les possibilités de mot de passe à travers des combinaisons de lettres, nombres et caractères spéciaux. À chaque mot de passe généré, le hash du mot de passe sera comparée avec le hash du mot de passe qu'on cherche à déchiffrer et si les 2 correspondent, alors le mot de passe a été déchiffré.

La troisième attaque est l'attaque par table arc-en-ciel (Rainbow Table Attack). Cette attaque va au début hacher une liste potentielle de mot de passe et stocker les mots de passe de texte claire et le hash de ces mots de passe dans une table. Une étape de réduction est ensuite réalisée où les hashes des mots de passe seront re-hachés et ces nouvelles valeurs de hash seront re-hachées aussi. Cette étape va être réalisée sur de nombreuses itérations pour générer de nombreuses chaînes de hash. L'attaquant va ensuite mettre le hash du mot de passe qu'il cherche à déchiffrer et une recherche inversée va être réalisée sur les nombreuses chaînes de hash générées en reculant jusqu'à trouver la bonne chaîne de hash. La dernière étape consiste juste à voir le texte clair représentant le hash trouvé qui correspond au mot de de passe déchiffré.

Source: <https://www.rapid7.com/fundamentals/brute-force-and-dictionary-attacks/>

Source: <https://nordvpn.com/blog/what-is-rainbow-table-attack/>

4.10. Choix des mots de passe

1.

```
(kali㉿kali)-[~/Documents]
$ echo 2072799-2061988 `date`
2072799-2061988 Wed Oct 11 11:40:38 PM EDT 2023

(kali㉿kali)-[~/Documents]
$ john --wordlist=rockyou.txt Password_File.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (sha512crypt, crypt(3) $6$ [S
HA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123456789          (simple)
sunshine           (brian)
monkey             (action)
liverpool          (vladimir)
4g 0:00:00:00 DONE (2023-10-11 23:40) 10.25g/s 656.4p/s 2625c/s 2625C/s 12345
6..freedom
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

(kali㉿kali)-[~/Documents]
$
```

Voici un tableau montrant le mot de passe correspondant à chaque utilisateur:

Utilisateur	Mot de passe
simple	123456789
brian	sunshine
action	monkey
vladimir	liverpool

2. Il est conseillé de ne pas utiliser le même mot de passe partout, car ceci augmente de beaucoup le risque d'avoir tous les comptes utilisés dans des applications compromises. Si le mot de passe a été utilisé dans une application

et que cette application subit une attaque avec la liste de mots de passe dévoilée, le pirate informatique pourra trouver le mot de passe utilisé associé à chaque utilisateur. Avec ce mot de passe en main et le nom d'utilisateur ou adresse email utilisé pour le compte, le pirate pourra trouver les autres applications utilisant ce nom d'utilisateur ou adresse email et s'authentifier à ces applications avec le même mot de passe trouvé. Cela lui permettra d'accéder à tous les comptes utilisés par une personne, et selon le type de compte, peut faire de nombreux dommages à la vie de la personne. On peut aussi faire un lien avec l'entropie. En utilisant le même mot de passe dans tous les comptes, on diminue l'entropie dans la gestion des comptes de la personne, car il n'y a pas d'aléatoire et d'imprévisibilité dans les mots de passe de ces comptes et, il suffira qu'une application ait une attaque ou une brèche informatique sur les mots de passe afin que tous les comptes d'une personne soient compromis.

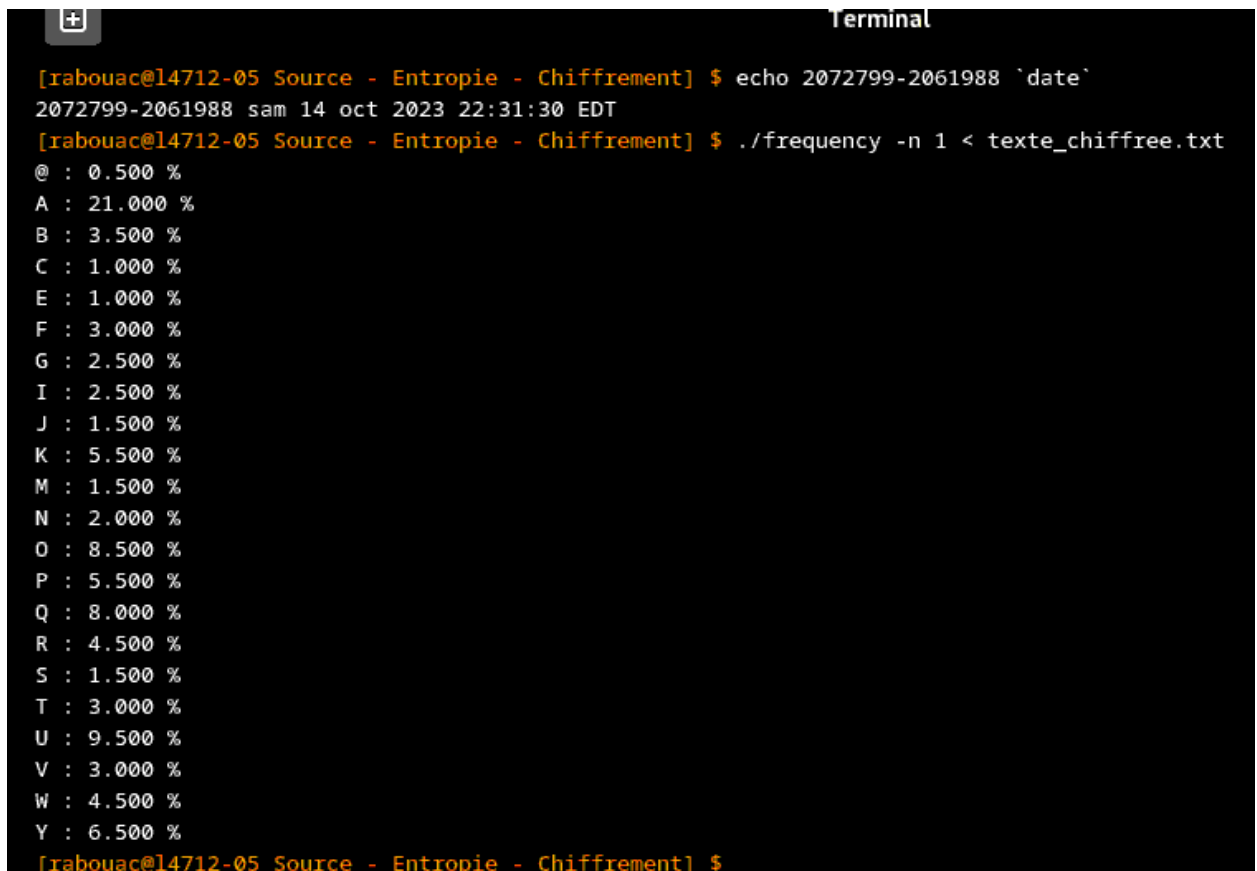
Un exemple de situation réelle serait une personne ayant 3 comptes différents qui utilisent le même mot de passe : le premier compte est un compte de messagerie Gmail, le deuxième compte est un compte dans un réseau social comme TikTok et le troisième compte est un compte dans une plateforme d'échange de crypto-monnaies comme Binance. Disons qu'une brèche de sécurité vient de se réaliser sur TikTok et que plusieurs mots de passe, dont le mot de passe de cette personne, sont compromis et dévoilés. Un attaquant, ayant accès à cette liste composée des adresses email et mots de passe des utilisateurs, pourrait trouver les comptes associés à cette personne dans différentes applications et essayer de s'y connecter. Il va donc réussir à se connecter au compte de messagerie gmail de cette personne du premier coup sans aucun effort et aura accès aux informations confidentielles et messages privés de la personne. Lorsqu'il essaye de se connecter à la plateforme d'échange de crypto-monnaie, il arrive à se connecter mais celle-ci utilise une double authentification et lui demande de vérifier son adresse email. L'attaquant ayant accès au compte de messagerie de la personne pourra vérifier la connexion et s'authentifier avec succès simplement en connaissant un seul mot de passe. Il va ensuite voir la balance monétaire de cette personne et pourra transférer toutes les crypto-monnaies possédées par la personne vers son portefeuille numérique. La balance monétaire de la personne sera vidée et elle aura perdu toutes ses économies dû à l'utilisation du même mot de passe dans plusieurs comptes.

3. Le lien serait que malgré que l'on utilise le même mot de passe et que ce mot de passe soit haché de manière sécuritaire dans la base de données des applications, il serait encore possible que tous les comptes d'une personne

soient compromis si l'attaquant a accès à un mot de passe haché. À l'aide des différents moyens possibles pour déchiffrer un mot de passe haché tels que vu en 4.9.6 avec l'attaque par force brute, par dictionnaire ou par table arc-en-ciel, l'attaquant pourra trouver le mot de passe en clair que la personne a utilisé pour s'authentifier et ainsi l'utiliser pour se connecter à tous les comptes de la personne utilisant le même mot de passe partout.

4.11. Déchiffrement simple

1)



```
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ echo 2072799-2061988 `date`
2072799-2061988 sam 14 oct 2023 22:31:30 EDT
[rabouac@l4712-05 Source - Entropie - Chiffrement] $ ./frequency -n 1 < texte_chiffree.txt
@ : 0.500 %
A : 21.000 %
B : 3.500 %
C : 1.000 %
E : 1.000 %
F : 3.000 %
G : 2.500 %
I : 2.500 %
J : 1.500 %
K : 5.500 %
M : 1.500 %
N : 2.000 %
O : 8.500 %
P : 5.500 %
Q : 8.000 %
R : 4.500 %
S : 1.500 %
T : 3.000 %
U : 9.500 %
V : 3.000 %
W : 4.500 %
Y : 6.500 %
[rabouac@l4712-05 Source - Entropie - Chiffrement] $
```

- 2) Le texte chiffré utilisé correspond au texte du chiffré du matricule 2072799 trouvé dans "INF4420A_TP1.csv" qui est le suivant:
- "OVUBAKPTTAAWQAEPJAKPMUASUUYARUNBUQAOYNUARPWFAQKOB
WYASGQAKOVAFOAVUACYOVAQKPQAWQAWRARUNBUQAPYJATOYIUBA
OTFAREPGIAKPFATWMUFAQKUBUATOYIAUYOGIKAYOVAQQA@WYFAOGQ
APYJQKWYIAQKUBUAWRAQOACYOVAPSOGQAQKORUANPMURAA"

Voici le texte déchiffré obtenu: "OWER HALLS IT MAY HAVE BEEN SECRET
ONCE SAID THORIN BUT HOW DO WE KNOW THAT IT IS SECRET ANY

LONGER OLD SMAUG HAD LIVED THERE LONG ENOUGH NOW TO FIND
OUT ANYTHING THERE IS TO KNOW ABOUT THOSE CAVES”

Démarche:

1. Notre première étape a été de trouver le caractère espace (“ ”). Après avoir utilisé l'utilitaire “frequency” avec une taille de 1, on trouve que le caractère “A” est le caractère le plus fréquent de la chaîne avec une apparition de 21%. C’est le seul caractère qui dépasse les 10% parmi tous les caractères. Sachant que dans les informations fournies dans l’énoncé de laboratoire, on sait que “l’espace est 1.07 fois plus fréquent que la lettre “e” en anglais. La lettre “E” ayant un pourcentage de fréquence de 12.702 % selon la table de la figure 4, elle représente la lettre ayant la plus grande fréquence parmi les lettres en anglais. Donc, notre lettre A a de fortes chances d’être l’espace. On sait que “une fin de phrase est représentée par deux espaces dans le texte original”, donc on fait l’analyse de fréquence “frequency” avec une taille de 2 pour trouver des paires de mêmes lettres qui pourraient former l’espace des fins de phrases. On obtient les paires “AA” avec une fréquence de 1.005%, “TT” et “UU” avec une fréquence de 0.503%. Sachant que “A” a de fortes chances d’être l’espace et que la paire “AA” est celle ayant la plus grande fréquence parmi les paires de lettres. On suppose alors que “A” représente l’espace.

2. En faisant “frequency” avec une taille de 1, on va trier la fréquence des lettres par ordre décroissant selon la table de fréquence de la figure 4. Si 2 lettres chiffrées ont la même fréquence, on va juste prendre les prochaines lettres dans la table et les assigner. Sans compter la lettre “A” qui est l’espace, les trois lettres ayant de grandes fréquences par rapport au reste des lettres sont U(9.5%), O(8.5%) et Q(8%). On veut vérifier si elles correspondent aux trois lettres “E”, “T”, “A”. En faisant “frequency” sur une taille de 2, on remarque que la seule paire de même lettre apparaît dans la lettre “U” avec UU (0.503 %). Parmi les lettres E, T ou A, la combinaison d’avoir une paire de mêmes lettres est très fréquente dans la lettre “E” avec “ee” qui est présente dans beaucoup de mots comme “see” ou “tree”. On va donc faire l’hypothèse que la lettre chiffrée “U” représente “E”. En connaissant “E”, on va essayer de trouver laquelle des lettres représente le “T” en essayant de trouver le digramme “th” et “he” qui sont les plus connus et les trigrammes “the” et “tha”. En faisant “frequency” avec une taille de 2, la lettre “Q” a deux digrammes de 2 lettres qui sont QK (3.015 %) et QO (1.005 %) alors que “O” en a plusieurs (OB : 0.503 %, OG : 1.508 %, OR : 0.503 %, OT : 0.503 %, OV : 2.513 %, OY : 1.508 %). Le digramme QK possède

une grande fréquence et ceci peut faire penser au diagramme "th". On fait alors "frequency" avec une taille de 3 et on obtient trois trigrammes intéressants pour "Q" qui sont : (QKO : 1.010 %, QKP : 0.505 %, QKU : 1.010 %, QKW : 0.505 %).

On remarque que les trigrammes commençant par QK ont une bonne fréquence et sont fréquents. Vu que U vaut E, le trigramme QKU pourrait probablement être THE. On vérifie les trigrammes commençant par O et on remarque qu'il n'y a pas de trigramme commençant par OK qui pourrait représenter TH et les trigrammes finissant par U qui représente E ont une fréquence de 0.505%. On peut alors déduire que "U" représente "E", "Q" représente "T" et "K" représente "H". Pour le "A", il faudrait d'autres vérifications car il est proche d'autres lettres comme O ou I.

3. En déchiffrant la phrase avec la table de correspondance réalisée jusqu'à maintenant, on peut trouver des mots qui commencent à se former comme "THERE IS", "IT", "IT IS SEWRET". On peut donc faire l'hypothèse que la lettre R représente bien S qui était là initialement en ordonnant les fréquences des lettres chiffrées avec la table de la figure 4. La lettre "W" chiffrée représentant la lettre "I" a été trouvée après avoir permuté avec la lettre "H" déchiffrée qui avait une légère différence dans leur fréquence. Avec la phrase formée "IT IS SEWRET", on suppose que le mot voulu est SECRET car on est sûr de toutes les lettres de cette phrase sauf pour le W qui représentait la lettre N chiffrée et le C qui représentait le W chiffré. On va donc les permuter pour avoir le "N" chiffré qui représente la lettre "C".

4. Après avoir permuté certaines lettres lors des changements, on se retrouve avec "O" représentant "A" qui avait été déduit à l'étape 2 et la lettre "P" chiffrée qui représente la lettre "O". On veut donc vérifier ces lettres car elles sont proches en termes de fréquence qui est élevée avec 8.167% pour A et 7.507% pour O selon la table de la Figure 4 de l'énoncé. Pour les différencier, on va comparer la fréquence du diagramme "at" car on sait que "ot" n'a pas une grande fréquence comparé à "at" qui fait partie des diagrammes les plus courants en anglais. On essaye les 2 combinaisons "OQ" et "PQ" car on sait que Q représente "T" trouvé précédemment. On trouve PQ avec une fréquence de 0.503% mais OQ n'est pas trouvable. On peut donc supposer que P chiffré représente "A" et O chiffrée représente "O". On peut confirmer cette hypothèse car on remarque que des mots en anglais et des suites de phrases qui font du sens ont été déchiffrées comme "HOW DO WE PNOW THAT IT IS SECRET" ou "ONCE SAID".

5. Avec la phrase déchiffrée “HOW DO WE PNOW THAT IT IS SECRET”, on suppose que le P devrait être un K qui ferait du sens dans la phrase avec le verbe KNOW. La lettre C chiffrée était celle qui représentait P, on va donc permuter les lettres P et K et avoir la la lettre C qui représente la lettre K déchiffrée.

6. Avec le texte qui commence à être déchiffré et faire du sens, on trouve des lettres déchiffrées qui doivent être modifiées. On peut remarquer dans cette phrase déchiffrée “SBAUM HAD LIYED THERE LONM ENOUMH NOW TO VIND OUT” que certaines lettres sont mauvaises et devraient être remplacées en comprenant le sens de la phrase. La première est la lettre Y qui devrait être un V. La lettre chiffrée M sera donc la lettre V. Ensuite “LONM ENOUMH” devrait signifier “LONG ENOUGH”, ce qui signifie qu’on devrait remplacer la lettre M par G. M correspond à la lettre chiffrée I et on va donc remplacer ça pour avoir la lettre chiffrée I qui représente G. Le même principe sera appliqué pour “NOW TO VIND OUT” qui devrait signifier “NOW TO FIND OUT” avec le V qui serait un F. Le caractère @ chiffré correspondra donc à la lettre F.

7. On utilise le même principe avec la phrase déchiffrée “TO FIND OUT ANGTHING THERE” où le G devrait être un Y. La lettre J chiffrée correspond alors à la lettre Y.

8. On applique ce principe une dernière fois sur cette suite de mots “IT BAY HAVE BEEN SECRET” où le “B” devrait être un M pour avoir le verbe “MAY”. On modifie alors dans notre table de correspondance la lettre “E” chiffrée qui va correspondre maintenant à la lettre “M”.

9. Finalement, on trouve le texte déchiffré suivant : “OWER HALLS IT MAY HAVE BEEN SECRET ONCE SAID THORIN BUT HOW DO WE KNOW THAT IT IS SECRET ANY LONGER OLD SMAUG HAD LIVED THERE LONG ENOUGH NOW TO FIND OUT ANYTHING THERE IS TO KNOW ABOUT THOSE CAVES” qui vient de “The Hobbit”.

Pour faciliter la manière de déchiffrer le texte à chaque étape ou essai, un code python a été fait avec la table de correspondance des lettres et le déchiffrement du texte :

```

decypher_text.py > ...
1
2 substitution_table = {
3     "A": " ", "B": "R", "C": "K", "D": " ", "E": "M",
4     "F": "D", "G": "U", "H": " ", "I": "G", "J": "Y",
5     "K": "H", "L": " ", "M": "V", "N": "C", "O": "O",
6     "P": "A", "Q": "I", "R": "S", "S": "B", "T": "L",
7     "U": "E", "V": "W", "W": "I", "X": " ", "Y": "N",
8     "Z": " ", "@": "F"
9 }
10 letters =
    "OVUBAKPTTTRAAWQAEPJAKPMJASUUYARUNBUQAQYNJARPWF AQKOBWYASGOAKOVAFQAVUJACYOVAQKPQAWQAMRARUNBUQAPYJATOYIUBAOTFAREPGIAKPFATWUMUFAQKUBUATQYIAUYOGIKAYO
    VAQQA@MYFAQGOAPYJQKWIQAQUBUAWRAQOACYOVAPSOGQAQKORUANPMURAA" "OVUBAKPTTTRAAWQAEPJAKPMJASUUYARUNBUQAQYNJARPWF AQKOBWYASGOAKOVAFQAVUJACYOVAQKPQAW
11
12 real_text = ""
13 for letter in letters:
14     real_text += substitution_table[letter]
15 print(letters)
16 print(real_text)

```