

RAPPORT TP1 INF4420A

Elliot Sisteron (1807165)

Partie A

Question 1

a) On cherche à calculer l'entropie d'un texte anglais de 200 caractères. Pour ce faire, on exécute la commande suivante dans le terminal :

```
./texte 200 | ./h-lettre
```

Ce qui retourne la sortie suivante :

```
(space) = 33
A = 5
B = 3
C = 5
D = 10
E = 23
F = 2
G = 1
H = 7
I = 16
J = 0
K = 1
L = 8
M = 7
N = 12
O = 16
P = 5
Q = 0
R = 7
S = 12
T = 15
U = 5
V = 0
W = 2
X = 0
Y = 4
Z = 1
Nombre total de caracteres : 200
Entropie de l'entree : 4.031273
```

FIGURE 1 – Entropie d'un texte anglais de 200 caractères

b) Cette entropie est proche de 4 bits. D'après le théorème de Shannon, un code compresseur à efficacité maximale nécessite donc 4 bits pour coder une des 26 lettres de l'alphabet anglais (en majuscule) et l'espace. Il existe avec certitude un code compresseur sans erreurs avec efficacité de 5 bits pour coder ces 27 caractères. En moyenne, on aura besoin de 4 bits pour coder individuellement chaque caractère.

c) Considérons maintenant le cas où ces lettres sont tirées au hasard de manière uniforme.

Notons S la v.a. associée, alors :

$$H(S) = \sum_{i=1}^{27} \frac{1}{27} \log_2 \left(\frac{1}{\frac{1}{27}} \right) = \log_2(27) \approx 4.75$$

d) Cette valeur représente l'entropie maximale atteignable pour 27 caractères. Ainsi, si l'on divise la valeur calculée à l'aide de notre programme par cette entropie maximale, cela représente le taux de compression (en codant chaque lettre individuellement) :

$$\tau = \frac{4.03}{4.75} \approx 84.85\%$$

e) Refaisons la même opération sur un texte généré de manière aléatoire en suivant la distribution des lettres dans un texte anglais.

`./lettre 200 | ./h-lettre`

Voici les résultats obtenus :

```
(space) = 29
A = 12
B = 1
C = 3
D = 7
E = 24
F = 3
G = 1
H = 9
I = 14
J = 0
K = 1
L = 7
M = 7
N = 12
O = 10
P = 3
Q = 0
R = 9
S = 16
T = 15
U = 2
V = 2
W = 7
X = 0
Y = 6
Z = 0
Nombre total de caracteres : 200
Entropie de l'entree : 4.066235
```

FIGURE 2 – Entropie d'un texte aléatoire (distribution des lettres en anglais) de 200 caractères

f) Les deux valeurs sont très proche, et c'est un peu surprenant car le texte généré avec *lettre* ne veut absolument rien dire (différence non significative, < 0.4). D'après la loi des grands nombre, ces deux valeurs vont même devenir égales lorsque la taille du texte généré augmente.

En réalité, l'entropie calculée ici et celle calculée précédemment ne se basent que sur la distribution des lettres et ne tiennent pas en compte des corrélations entre les lettres (digrammes, trigrammes, ...) et les mots. Si c'était le cas, la différence serait très significative.

Question 2

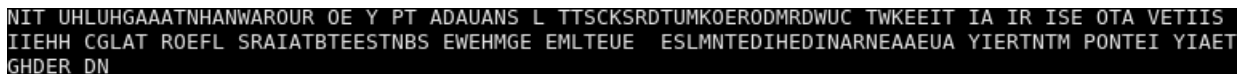
- a) Commençons par générer deux textes : un avec *texte* et un avec *lettre*.

```
./texte 200 > texte.txt  
./lettre 200 > lettre.txt
```



TANDS THE LEAGUE BETWEEN THE SCOT AND VS MO CRACKT AND DISSEUERED MY RENOWNED LORD THE TREACHE
ROUS KING NO SOONER WAS INFORMDE OF YOUR WITH DRAWING OF YOUR ARMY BACKE BUT STRAIGHT FORGETTING
OF HIS

FIGURE 3 – Texte de 200 caractères généré avec *texte*

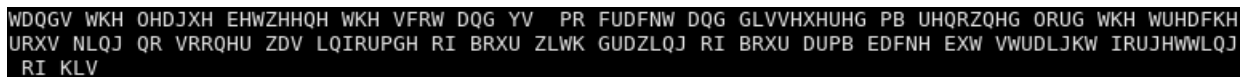


NIT UHLUHGAATNHNANWAROUR OE Y PT ADAUANS L TTSCSRDTUMKOERODMRDWUC TWKEEIT IA IR ISE OTA VETIIS
IIEHH CGLAT ROEFL SRAIATBTTEESTNBS EWEHMG EMLTEUE ESLMNTEDIHEDINARNEAAEUA YIERTNTM PONTEI YIAET
GHDER DN

FIGURE 4 – Texte de 200 caractères généré avec *lettre*

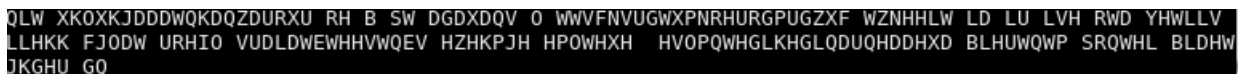
Nous allons maintenant chiffrer ces textes.

```
./cesar < texte.txt > texte_cesar.txt  
./cesar < lettre.txt > lettre_cesar.txt
```



WDQGV WKH OHDJXH EHWZHHQH WKH VFRW DQG YV PR FUDFNW DQG GLVVHXHUHG PB UHQZQHG ORUG WKH WUHDFKH
URXV NLQJ QR VRRQHU ZDV LQIRUPGH RI BRXU ZLWK GUDZLQJ RI BRXU DUPB EDFNH EXW VWUDLJKW IRUJHWWLQJ
RI KLV

FIGURE 5 – Texte généré avec *texte* chiffré avec César



QLW XKOKXKJDDDWQKDQZDURXU RH B SW DGDxDQV O WWVFNVUGWXPNRHURGPUGZXF WZNHHLW LD LU LVH RWD YHWLLV
LLHKK FJODW URHIO VUDLDWEWHHVWQEV HZHKPJH HPOWHXH HVOPQWHGLKHGLQDUQHDDHXD BLHUWQWP SRQWHL BLDHW
JKGHU GQ

FIGURE 6 – Texte généré avec *lettre* chiffré avec César

Pour les déchiffrer, il suffit d'utiliser la commande suivante :

```
./cesar-d < texte_cesar.txt  
./cesar-d < lettre_cesar.txt
```

On retrouve bien les originaux.

- b) Nous allons maintenant trouver les fréquences des lettres dans chacun de ces textes avec *h-lettre* et les mettre sous forme d'histogramme.

Voici les résultats :

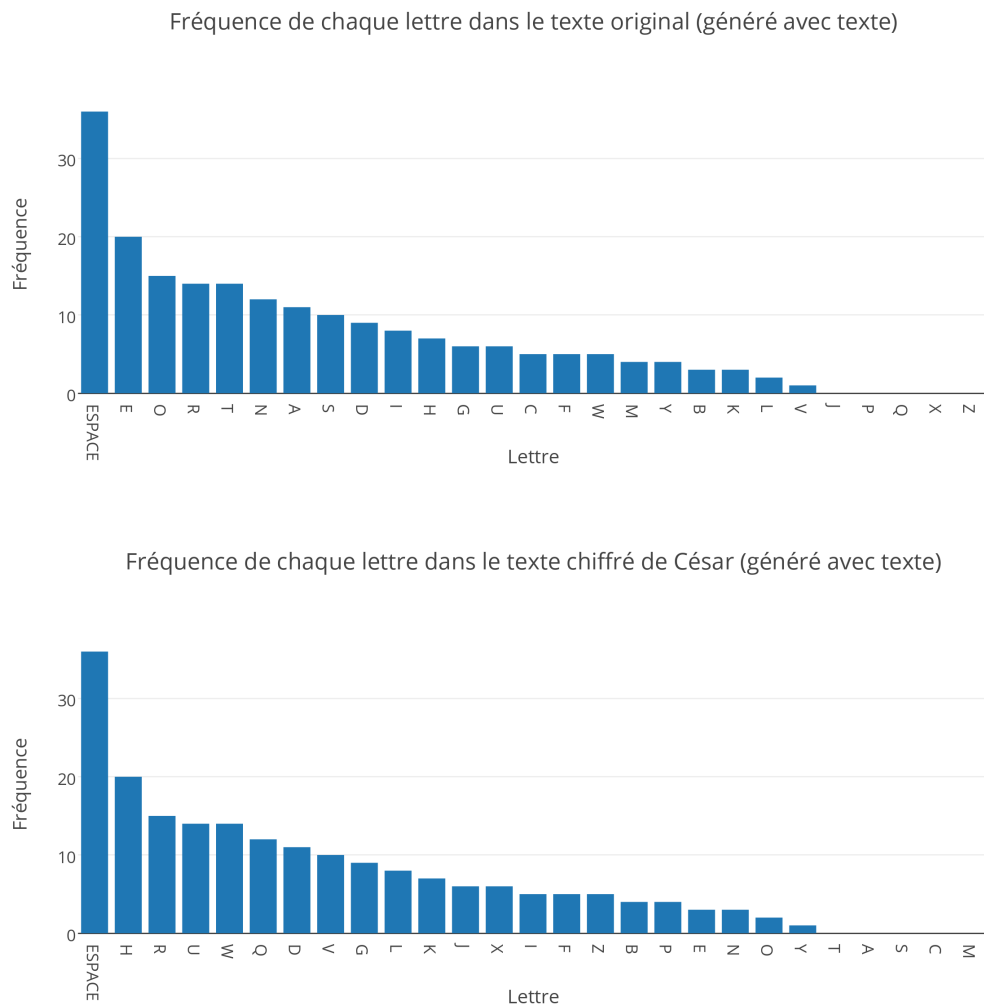


FIGURE 7 – Histogrammes pour le texte généré avec *texte*



FIGURE 8 – Histogrammes pour le texte généré avec *lettre*

c) Première remarque : le chiffre de César utilisé ici ne chiffre pas les espaces. En effet, leur proportion reste identique dans les deux textes. Deuxième remarque : les histogrammes clairs et chiffrés sont identiques une fois triés. C'est tout à fait normal car le chiffre de César est dit mono-alphabétique : il associe de manière bijective une lettre à une autre. Enfin, on peut facilement déduire des histogrammes que la clé du chiffre de César utilisé est ici un simple décalage de 3 lettres vers la gauche (le E devient H, le F devient I, ...). En plus d'être sensible à une attaque brute-force (25 possibilités à tester), on constate que le chiffre de César tombe à la moindre analyse fréquentielle.

À quoi ressemblerait ces histogrammes pour les digrammes ? Regardons le cas de *ee* : dans un texte anglais, il aura de forte chance d'apparaître ("tee-shirt", "flee", "bruce lee", "fee", ...). De même pour *th* ("the", "thirsty", "third", ... 3.7% de chance d'apparition en anglais). Dans un texte généré de manière aléatoire avec la distribution de la langue anglaise, chaque digramme aura une fréquence qui est la multiplication des probabilités associées aux deux lettres. Dans un vrai texte en anglais, la probabilité de voir une lettre apparaître dépendra directement des autres lettres du texte. Dans un texte généré de manière aléatoire, cette dépendance n'existe pas.

d) Peut-on utiliser cette nouvelle information pour faire une attaque efficace contre le chiffre de César ? Cela nous aidera pour *texte* mais certainement pas pour *lettre*. En effet, dans le cas de *texte*, si l'on pose S la v.a. génératrice, on devrait avoir $H(S^2) < 2H(S)$; i.e. $\frac{H(S^2)}{2} < H(S)$ et donc un taux de compression plus faible (ainsi il sera plus simple de déchiffrer le texte).

Dans le cas de *lettre*, l'entropie double et donc le taux de compression reste le même ; il n'y a aucun gain à procéder ainsi. On peut donc procéder par brute-force ou raisonner sur les fréquences de lettres pour casser le chiffre. Toutefois, le texte déchiffré ne voudra toujours rien dire vu qu'il a été généré aléatoirement.

Question 3

a) On génère deux fichiers contenant les résultats de l'exécution de *monnaie* et *binaire* sur 1024 octets.

```
./monnaie 1024 > monnaie.bin
./binaire 1024 > binaire.bin
```

Voici les résultats de l'exécution de *h-bit* et *h-ascii* sur ces deux fichiers :

<pre>0 = 4157 1 = 4035 Nombre total de bits : 8192 Entropie du texte entre : 0.999840</pre>	<pre>Nombre total d'octets : 1024 Entropie de l'entree : 7.793288</pre>
---	---

FIGURE 9 – Les résultats de h-bit et h-ascii sur monnaie.bin

<pre>0 = 5176 1 = 3016 Nombre total de bits : 8192 Entropie du texte entre : 0.949252</pre>	<pre>Nombre total d'octets : 1024 Entropie de l'entree : 0.788881</pre>
---	---

FIGURE 10 – Les résultats de h-bit et h-ascii sur binaire.bin

L'entropie de *monnaie* est très proche de 1 et c'est tout à fait normal : il s'agit d'une distribution uniforme et donc l'entropie doit être maximale. De même pour l'entropie par caractère ASCII, elle est proche de 8.

Pour *binaire*, l'entropie par bit est élevée et on pourrait être tenté de penser que les bits sont générés uniformément aussi. Or, l'entropie par caractère est extrêmement faible (0.78), ce qui signifie qu'il y a une forte dépendance entre les bits et que cet algorithme secret génère probablement une chaîne ASCII intéressante.

En effet, le fichier *binaire.bin* généré ne contient que deux caractères « { » et « . ». Ces deux caractères sont complémentaires en nombre de 0 et de 1, d'où l'entropie par bit élevée.

b) On génère maintenant une clé via *monnaie* et on l'applique aux fichiers *monnaie.bin* et *binaire.bin*.

```
./monnaie 1024 > masque.bin
./masque masque.bin 1024 monnaie.bin monnaie_chiffre.bin
./masque masque.bin 1024 binaire.bin binaire_chiffre.bin
```

Voici les résultats de l'exécution de *h-bit* et *h-ascii* sur ces deux nouveaux fichiers :

```
0 = 4066
1 = 4126
Nombre total de bits : 8192
Entropie du texte entre : 0.999961
```

```
Nombre total d'octets : 1024
Entropie de l'entree : 7.813127
```

FIGURE 11 – Les résultats de h-bit et h-ascii sur monnaie_chiffre.bin

```
0 = 4061
1 = 4131
Nombre total de bits : 8192
Entropie du texte entre : 0.999947
```

```
Nombre total d'octets : 1024
Entropie de l'entree : 7.817150
```

FIGURE 12 – Les résultats de h-bit et h-ascii sur binaire_chiffre.bin

Là encore, l'entropie par bit est très élevée. L'entropie par caractère ASCII passe à 7.81 pour les deux fichiers.

c) La méthode du masque jetable est ainsi sécuritaire : l'entropie devient très forte ; si la clé est totalement aléatoire (comme ici), il est impossible de déchiffrer le message. Mais, cela nécessite d'avoir une clé de la taille du message et à usage unique, ce qui est très contraignant.

Question 4

a) L'entreprise PokerMaxProUltime est une compagnie de poker sur internet. On se doute bien que ces parties se jouent en temps réel ; il est donc absolument nécessaire d'avoir un site fiable. Une coupure non attendue du site web pourrait causer de gros dommages étant donné que les joueurs mettent de l'argent en jeu sur le site ; il y a une perte à gagner pendant les réparations. Les joueurs qui dépensent beaucoup sur ce site seraient susceptibles de partir chez un concurrent.

On va estimer que l'impact d'un ouragan est de 300 000\$ (perte à gagner estimée à 50 000\$, perte des clients estimée à 50 000\$, coût des réparations de 100 000\$ et 100 000\$ d'argent investi dans l'installation perdu). L'espérance de perte par an est donc :

$$E = 25\% \times 300\,000 = 75\,000\$$$

Autrement dit, on perdrait en moyenne 75 000\$ par an, ce qui n'est plus rentable à partir de 6 ans et demi environ. Si l'entreprise compte s'installer aux Caraïbes sur le long terme, il est clairement plus intéressant de payer plus cher pour un emplacement moins risqué.

b) Passons à l'analyse des trois scénarios donnés :

1. Dans ce scénario, le malfaiteur touche à l'intégrité de notre système car les données sont altérées pour lui permettre de tricher.
2. Ici, le malfaiteur inonde notre serveur de requêtes et donc il touche à la disponibilité du système.
3. Dans ce cas, le malfaiteur touche à la confidentialité du système car il va accéder à des données qui sont supposées être confidentielles.

c) Complétons maintenant l'étude de risque de notre patron pour trois scénarios. La capacité, l'opportunité, la motivation et l'impact sont notés sur une échelle de 1 à 4. Pour calculer les probabilités, on choisit de prendre la moyenne de la capacité, l'opportunité et la motivation. Le risque n'est que la probabilité de l'incident multipliée par l'impact.

1. Les tricheurs représentent le plus gros risque.

Acteur	Capacité	Opportunité	Motivation	Probabilité	Impact	Risque
Tricheur	4	4	4	4	2	8
C.O.	1	4	1	2	2	4
Concurrents	2	4	2	2.66	2	5.32

2. Les concurrents sont les acteurs les plus dangereux.

Acteur	Capacité	Opportunité	Motivation	Probabilité	Impact	Risque
Tricheur	1	4	1	2	4	8
C.O.	4	4	1	3	4	12
Concurrents	2	4	4	3.33	4	13.32

3. Le crime organisé représente le plus gros risque.

Acteur	Capacité	Opportunité	Motivation	Probabilité	Impact	Risque
Tricheur	1	3	1	1.66	3	5
C.O.	4	3	4	3.66	3	11
Concurrents	1	3	2	2	3	6

d) Regardons trois nouveaux scénarios et étudions l'impact qu'ils auraient sur les paramètres et le risque.

1. Dans ce scénario, les concurrents peuvent être amenés à nous attaquer suite à notre succès. La motivation des concurrents est donc bien plus élevée. La capacité peut éventuellement diminuer en considérant que les concurrents sont en faillite.
2. Ici, la mafia voudra faire un exemple pour imposer son régime de terreur. Pour ce faire, il y a fort à parier qu'elle investira grandement pour montrer sa puissance. Les facteurs de capacité et de motivation augmenteront.
3. Dans ce cas, la capacité des tricheurs ne va pas augmenter. Notre nouveau système diminuera grandement l'opportunité. Cela pourra éventuellement avoir pour conséquence de diminuer la motivation des tricheurs (le risque de se faire attraper augmentant, il est moins intéressant de tricher).

e) Dans le cas du vendeur de service de surveillance, la grille de risque du troisième scénario devient :

Acteur	Capacité	Opportunité	Motivation	Probabilité	Impact	Risque
Tricheur	1	3	1	1.66	3	5
C.O.	4	4	4	4	4	16
Concurrents	1	1	2	1.33	3	4

On remarque deux choses : ce nouveau service fera chuter l'opportunité des concurrents car nous serons protégés des intrusions. Mais, on peut assumer que la mafia pourra facilement soudoyer le fournisseur situé en Russie. Si c'est le cas, ces malfaiteurs pourront avoir un impact dévastateur sur notre système étant donné que le fournisseur peut prendre le contrôle à distance. De plus, ce prix si bas est étrange et nous devrions nous méfier.

Nous avons donc réduit un peu le risque des concurrents (de 2 points) et grandement augmenté le risque des mafieux (de 4 points). Le jeu n'en vaut pas la chandelle.

Partie B

Question 1

a) Les alphabets utilisés ici sont les suivants :

— $\sigma = [0 - 9], |\sigma| = 10$

— $\tau = [0 - 1], |\tau| = 2$

— $\tau' = [0 - 1], |\tau'| = 2$

b) Les langages associés sont :

— $L_\sigma = \{ww \mid w \in [0 - 9]^4\}, |L_\sigma| = 10000$

— $L_\tau = \{ASCII(w) \mid w \in L_\sigma\}, |L_\tau| = 10000$ avec ASCII une fonction qui convertit tous les caractères d'un mot au format ASCII.

— $L_{\tau'} = [0 - 1]^{64}, |L_{\tau'}| = 2^{64}$

c) et d) Ce système est vulnérable à deux types d'attaques.

— Un attaquant pourrait tenter une simple attaque sur le message intercepté pour en obtenir le contenu clair. Au vu de la taille de la clé (56 bits), cela est faisable en un temps raisonnable avec le matériel d'aujourd'hui. De plus, cette clé n'est pas changée dans le système, ce qui fait que l'attaquant n'est pas limité par le temps. Les messages envoyés au système de chiffrement sont restreints du fait qu'il n'existe que 10000 NIP possibles. Or, il existe 2^{64} possibilités de messages dans $L_{\tau'}$. Donc, on peut vérifier rapidement si une clé est valide ou non en l'appliquant pour déchiffrer un message. En effet, si le message déchiffré avec la clé testée ne fait pas partie du langage L_τ , alors cette clé n'est pas valide. Prenons un exemple simple et lançons ce programme deux fois de suite :

```
python2.7 transBase.py 3451 | hexdump
```

On constate que les octets chiffrés sont bien les mêmes.

On peut même faire encore mieux et convertir chacun des NIP possible en son complément chiffré. De ce fait, on a une table associative et on peut facilement déchiffrer un message. Bien sûr, cette méthode n'est pas vraiment réaliste pour le hacker dans le sens où il peut difficilement envoyer 10000 NIP différents au système pour observer leur sortie.

— Il peut aussi brouiller les messages en modifiant leur contenu. Encore mieux, si il est client de cette banque (ou si un de ses amis l'est), il peut connaître un NIP clair et sa version chiffrée (en envoyant son NIP dans le système et en observant la sortie). Il lui suffit alors d'envoyer ce NIP chiffré au système pour remplacer le NIP du client par celui qu'il connaît. Faisons un essai, convertissons le NIP 1452 avec notre système, puis envoyons le message chiffré à la banque.

```
python2.7 transBase.py 1452 | python2.7 recepBase.py
```

On retrouve bien 1452.

e) Considérons maintenant les trois nouveaux codages proposés, ces trois codages prennent le NIP et l'encode sur 14 bits (l'entropie maximale pour coder 10000 NIP est d'environ $\log_2(10000)$, i.e. 14 bits), puis ajoute 2 bits de parité (pour l'intégrité du message, cela permet de connaître d'éventuelles modifications) :

1. Le premier codage est effectué avec un nombre aléatoire sur 48 bits, ce qui rend l'attaque

par force brute impossible. Un NIP ne sera jamais chiffré de la même manière du fait du caractère aléatoire de l'encodage. Toutefois, l'attaque d'envoi d'un NIP chiffré connu marche toujours.

En effet, l'exécution de ce programme deux fois de suite ne retourne plus la même suite de bits :

```
python2.7 trans1.py 5524 | hexdump
```

2. En plus du nombre aléatoire (maintenant sur 16 bits), ce second codage ajoute un timestamp sur 32 bits qui permet de vérifier que le message est récent et permet ainsi de contrer l'attaque d'envoi de NIP chiffré connu (ce NIP sera plus ancien, ce qui compromet l'attaque).

Faisons un essai :

```
python2.7 trans2.py 7209 > test.bin
```

Attendons quelques minutes et lançons cette commande :

```
python2.7 recep2.py < test.bin
```

Cela a pour résultat : « Délai de transmission suspect, opération annulée ».

3. Ici le nombre aléatoire disparaît pour laisser sa place à l'ancien NIP. C'est moins sécuritaire dans le sens où l'attaque par force brute devient plus réalisable, même si elle reste difficile. Mais, cela nous protège encore plus dans le cas d'envoi de NIP chiffré connu car même si le timestamp est bon il y a une vérification au niveau de l'ancien NIP.

f) Le meilleur codage semble ici être lié à l'efficacité du timestamp. Si le timestamp est fiable, on préférera le second car il permet de nous protéger de l'attaque par force brute de manière plus sécuritaire. Dans le cas inverse, on choisira plutôt le troisième car l'ancien NIP permet de mieux se protéger contre l'attaque par envoi de message.

Question 2

a) Lorsque l'on se connecte avec le faux site, Firefox refuse la connexion en spécifiant que ce n'est pas sécuritaire. La connexion est pourtant bien en *https*. Mais, si l'on regarde le certificat du site, on s'aperçoit qu'en réalité l'autorité l'ayant vérifié est le site lui-même ; ce qui entraîne ce message de Firefox.

b) Cela est très louche pour une banque et devrait nous alerter immédiatement que ce site est une fraude. Les noms d'organisation et de département sont aussi suspects (Hackers, Inc. et Phising Dept.). Si nous ne sommes pas convaincu, nous pouvons toujours appeler la banque pour vérifier.

c) Si l'on ajoute l'exception de sécurité à Firefox, la connexion peut s'établir (et un autre utilisateur de la même machine pourra même se rendre sur le site sans avertissement). Cette exception assure à Firefox que la connexion est sécuritaire (même si ce n'est pas le cas), il fera donc abstraction des failles de sécurité qu'il pensait avoir détecté.

d) On génère maintenant un certificat CA et un certificat pour *desjardins.com* (avec le certificat CA). On arrive à accéder au site sans problème. Cela vient du fait que le certificat du site est signé par un CA qui est ajouté à Firefox et qui est donc considéré comme fiable.

e) Connectons-nous aux sites <https://www.rbc.com> et <https://www.bmo.com>. Il y a un message d'erreur qui nous informe que l'identité de ces sites web ne peut pas être vérifiée du fait qu'ils sont signés par des CA inconnues.

f) On ajoute une exception temporaire pour le site de la bmo. On parvient donc à y accéder. Par contre, si on change de site web et que l'on efface le cache de Firefox, on ne peut plus y accéder. On en déduit donc que cette exception temporaire est probablement stockée dans le cache de Firefox ; il n'enregistre pas le certificat pour ne pas rendre cette exception permanente.

g) Faisons de même pour le site de la rbc. Une fenêtre apparaît nous demandant si l'on veut faire confiance à ce CA pour plusieurs cas. On coche les trois cases et l'on valide. Si l'on change de site et que l'on vide le cache, on peut encore revenir sur le site de la rbc car nous avons accepté ce CA.

h) En retournant sur le site de la bmo, on constate qu'on peut y accéder aussi : leur certificats sont signés par le même CA que l'on a considéré comme sûr.

i) Il est donc dangereux d'accepter des certificats self-signed car cela ne garantit pas l'identité du site web : il faut qu'il soit signé par une autorité de confiance. Cela cache généralement une tentative de phishing : le site en question se fait passer pour une entité connue pour récupérer des informations sensibles. L'ajout d'un certificat CA est aussi à éviter car tous les sites certifiés par ce même CA apparaîtront comme étant de confiance pour Firefox. Il n'y aura plus d'avertissements, et donc si ce CA signe des sites malveillants, l'utilisateur ne sera pas prévenu.

Question 3

a) On exécute la commande suivante pour chiffrer l'image avec AES en mode ECB :

```
python2.7 AES.py -i mdp.jpg -m ECB -o mdp_chiffre.jpg
```

Voici le résultat :

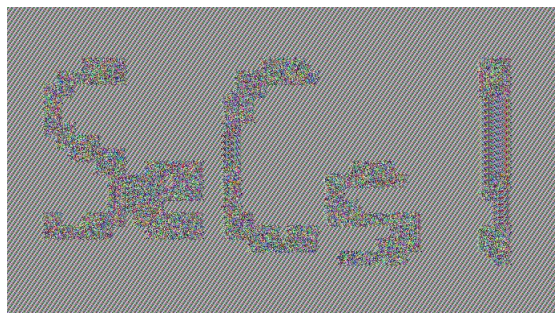


FIGURE 13 – Image contenant le mot de passe chiffrée avec AES en mode ECB

Le mot de passe est encore très lisible (SeCsl), ce n'est pas sécuritaire.

b) Utilisons maintenant le mode CBC :

```
python2.7 AES.py -i mdp.jpg -m CBC -o mdp_mieux_chiffre.jpg
```

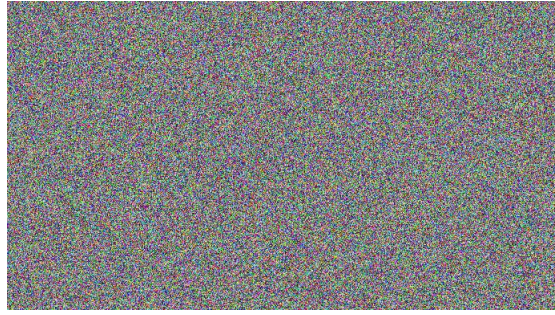


FIGURE 14 – Image contenant le mot de passe chiffrée avec AES en mode CBC

On constate que le mot de passe n'est plus lisible du tout, ce n'est que du bruit.

c) ECB va chiffrer des blocs identiques de la même manière, alors que CBC va diffuser le changement d'un bloc dans tout le message et complique la relation entre le message et la clé. C'est donc clairement la méthode à privilégier.

Question 4

a) Le fichier *passwd* ne contient pas de mot de passes, c'est assez surprenant vu le nom du fichier... Mais, quand on y pense, on peut y accéder en lecture et donc il est évident qu'on ne va pas y trouver les mots de passes des utilisateurs (même chiffrés). Ce fichier contient en réalité des informations sur les utilisateurs de la machine (identifiant, groupe(s) auquel(s) ils appartiennent...). Il est tout à fait normal de pouvoir accéder à ces informations car elles sont nécessaire au bon fonctionnement de l'OS : certains programmes ont besoin de connaître les permissions accordée à chaque utilisateur interagissant avec la machine. Par contre, personne ne peut modifier ce fichier (sauf l'utilisateur root, évidemment). C'est tout à fait normal car on ne voudrait pas que quelqu'un s'ajoute à ce fichier pour se donner des droits supplémentaires.

b) Ajoutons l'utilisateur « elliot » à la machine. Les deux fichiers (*passwd* et *shadow*) sont modifiés. Dans chaque fichier, on constate l'ajout d'une ligne avec le nom de l'utilisateur.

On lui donne un mot de passe qu'on ne divulguera pas ici (ce TP n'est en fait qu'une futile tentative de phishing, bien tenté). Seul le fichier *shadow* est maintenant modifié. C'est logique : le fichier *passwd* ne contient pas de mot de passe il n'a donc pas de raison d'être modifié. Par contre, le fichier *shadow* les contient du fait de sa modification : on constate qu'un « ! » vient d'être remplacé par une chaîne de caractères.

c) Là encore, la modification de mot de passe n'entraîne une modification que du fichier *shadow*. Ce fichier n'est accessible ni en écriture, ni en lecture (sauf par root, toujours). Il a donc toutes les chances de contenir les mots de passes des utilisateurs.

d) L'opération de remplacer le mot de passe par le même affecte encore le fichier *shadow*. On imagine que les mots de passes sont chiffrés dans *shadow*. Pour éviter que quelqu'un puisse déduire que deux utilisateurs ont le même mot de passe, on voit bien que deux mêmes mots de passes ne sont pas chiffrés de la même manière.

e) Créons un nouvel utilisateur « oscar » avec un mot de passe. On modifie son mot de passe dans *shadow* par celui du premier utilisateur. De manière surprenante, cela fonctionne. Cela vient de la façon sont hashé les mots de passes, il est normal que la fonction de hashage

renvoie le même mot de passe pour le même hashage. En réalité, il est normal pour l'utilisateur root d'avoir ce genre de permissions (après tout, il est le propriétaire de la machine, libre à lui de faire ce qu'il veut).

f) Comme on se l'imaginait, la suppression d'utilisateur enlève les lignes le concernant dans les fichiers *passwd* et *shadow*.

Question 5

a) John The Ripper trouve ici quatre mots de passes. Les hashes et les mots de passes sont listés sur la figure suivante :

On constate qu'il s'agit là de mot de passe assez faibles :

- Un utilisateur a utilisé son nom d'utilisateur comme mot de passe (claudia)... Un autre une variante (john1).
- On trouve aussi un simple mot (security).
- Plus surprenant, un mot de passe constitué de chiffre et de lettre sans trop de signification fait aussi partie du lot (0244fni).

b) Les entropies maximales sont les suivantes :

- Pour l'alphabet [a-zA-Z], il y a 52 éléments donc $\log_2(52) \approx 5.7$, i.e. 6 bits par symbole.
- Concernant [a-zA-Z0-9], $\log_2(62) \approx 5.95$, donc 6 bits aussi, le résultat trouvé en 1 devient donc moins surprenant.
- Pour la table ASCII, il y a 128 caractères, soit une entropie maximale de $\log_2(128) = 7$ bits.

c) Pour qu'un mot de passe soit fort, il ne faut pas tant qu'il ait des chiffres, mais surtout qu'il utilise de nombreux caractères, même ceux que l'on retrouve peu réellement comme % ou \$... Il faut utiliser de manière équiprobable les caractères de la table ASCII et donc éviter les répétitions.

d) Voici trois critères importants pour qu'un mot de passe soit difficilement attaquant par notre ennemi John.

- Éviter d'utiliser des mots ou de manière générale des choses qui ont un sens dans une langue. Il ne faut donc pas aussi qu'il soit proche du nom de l'utilisateur.
- Utiliser de nombreux caractères (voir surtout les moins fréquents, car ils sont rarement ciblés par les hackers).
- Répartir uniformément les caractères. Éviter les caractères fréquents et les répétitions.

Partie C

Question 1

a) Alice chiffre chaque caractère séparément avec la clé de Bob, ce qui est équivalent à faire une substitution mono-alphabétique car chaque lettre sera chiffrée de la même manière dans ce cas. Cette clé publique est connue de Ève. Il lui suffit donc de chiffrer chaque lettre de l'alphabet pour avoir une table associative qui fait correspondre chaque lettre en clair à son équivalent chiffré, ce qui lui permet de déchiffrer le message. Même si Ève n'avait pas accès à la clé publique, il lui suffirait d'une simple analyse fréquentielle pour casser le chiffre.

b) On récupère le texte à déchiffrer et la clé publique pour le matricule 1807165. On se fait une petite fonction en python qui nous génère la table associative :

```
>>> def print_l():
...     for i in range(26):
...         print(chr(ord('A')+i), (i*387)%294493)
...
>>> print_l()
A 0
B 1
C 85384
D 173737
E 253241
F 201179
G 158612
H 49072
I 170005
J 190641
K 280032
L 112165
M 77417
N 35556
O 211737
P 39725
Q 146950
R 205128
S 179555
T 22806
U 71125
V 49714
W 184000
X 259846
Y 277743
Z 228065
>>>
```

FIGURE 15 – Table de substitution pour $e = 387$ et $N = 294493$

Cela nous donne donc le mot « ceder ».

c) Les nombres 0 et 1 ne changent pas lorsqu'ils sont élevés à la puissance. Ainsi, peu importe les valeurs de N et de e , A et B seront toujours chiffré en A et B. Il faut donc éviter d'utiliser ces nombres.

On en conclut qu'il faut absolument utiliser un alphabet très grand pour contrer ce genre d'attaque. Il faudrait, par exemple, coder plusieurs caractères en même temps ou encore procéder de la même manière que dans la question 1 de la partie B en ajoutant des valeurs aléatoires.

Question 2

On récupère le texte chiffré pour le matricule 1807165. Après analyse fréquentielle, la lettre prédominante est le "F" (41 apparitions), c'est donc clairement l'espace car on la retrouve parfois trois fois à la suite (et c'est impossible en anglais). On fait une première attaque par recuit simulé en se basant sur un outil statistique qui est la *fitness* pour calculer le score d'une solution. Certaines lettres sont à modifier, mais le texte est déjà tellement bien déchiffré que les substitutions restantes sont évidentes.

Voici le texte déchiffré :

« NING SUCCESS MARY GAVE HIM A BRANDNEW BARLOW KNIFE WORTH TWELVE
AND A HALF CENTS AND THE CONVULSION OF DELIGHT THAT SWEPT HIS SYSTEM
SHOOK HIM TO HIS FOUNDATIONS TRUE THE KNIFE WOULD NOT CUT ANY »