



École Polytechnique de Montréal

Département de génie informatique et génie logiciel

Travail Pratique 1

INF8085

SOUMIS PAR :

Antoine Pichon, 2489005

Abdel Hamadouche, 2217813

Le 10/07/2025

Table des matières

1 Entropie et sources d'information	3
1.1 Question 1	3
1.2 Question 2	3
1.3 Question 3	3
1.4 Question 4	3
2 La librairie de Babel	3
2.1 Question 1 et 2	3
2.2 Question 3	4
2.3 Question 4	4
3 Histogrammes	4
3.1 Question 3	4
3.2 Question 5	5
3.3 Question 6	6
3.4 Question 7	7
4 Masque jetable	7
4.1 Question 1	7
4.2 Question 3	7
4.3 Question 4	8
5 Communication à clé publique, HTTPS et SSL	8
5.1 Question 1	8
5.2 Question 2	8
5.3 Question 3	9
5.4 Question 4	9
5.5 Question 5	9
5.6 Question 6	11
5.7 Question 7	12
6 Codage	13
6.1 Question 1	13
6.2 Question 2	13
6.3 Question 3	13
7 Changement de Codage	14
7.1 Question 1	14
7.2 Question 2	14
8 Chiffrement par bloc et modes d'opération	15
8.1 Question 1	15
8.2 Question 2	16
9 Organisation des mots de passe en UNIX/Linux	17
9.1 Question 1	17
9.2 Question 2	20
9.3 Question 3	21
9.4 Question 4	22
9.5 Question 5	23

9.6 Question 6	24
10 Choix des mots de passe	25
10.1 Question 1	25
10.2 Question 2	25
10.3 Question 3	25
11 Déchiffrement simple	26
11.1 Question 1	26
11.2 Question 2	26

1 Entropie et sources d'information

1.1 Question 1

Dans le cas d'un alphabet de 32 caractère, l'Entropie moyenne par caractère se calcul grace à la formule de Shannon :

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2(p(x_i))$$

Avec n le nombre de caractère dans l'alphabet et $p(x)$ la probabilité d'apparition du caractère x dans le texte. Dans notre cas, les probabilités sont toutes égales donc $p(x) = \frac{1}{32}$. Donc l'entropie moyenne par caractère est :

$$H(X) = - \sum_{i=1}^{32} \frac{1}{32} \log_2\left(\frac{1}{32}\right) = -32 * \frac{1}{32} * \log_2\left(\frac{1}{32}\right) = 5$$

Donc l'entropie moyenne par caractère est de 5 bits.

1.2 Question 2

Dans notre cas, la distribution est uniforme donc on peut dire que l'entropie maximale est atteinte.

1.3 Question 3

Il n'est pas possible de compresser un texte dont les caractères sont distribués uniformément. En effet, la compression de données repose sur la redondance de motifs or dans notre cas, il n'y a pas de redondance car les motifs sont tous générés aléatoirement indépendamment les uns des autres.

1.4 Question 4

Dans le cadre de la compression, plus le texte (ou l'image) à de redondance, plus il est possible de le compresser sans perdre d'information. Souvent, on trouve dans les textes (peu importe la langue) des motifs cohérent qui se répète et qui permettent de compresser les textes. Dans le cas des images, celle ci peuvent avoir des motifs géométrique ou de symétrie ou autres qui permettent de les compresser. On peut conclure en disant que plus l'entropie est faible, plus il est possible de compresser un texte, une image, ou autre sans perdre d'information

2 La librairie de Babel

2.1 Question 1 et 2

Nous avons calculé grace à l'executable "h-ascii" l'entropie des deux textes :

```
2489005-2217813 30-09-2025
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-ascii < texte1.txt
Nombre total d'octets : 3249
Entropie de l'entrée : 4.889230

2489005-2217813 30-09-2025
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-ascii < texte_total.txt
Nombre total d'octets : 6480
Entropie de l'entrée : 4.891331
```

Le premier terminal correspond au texte de la bibliothèque de babel de 3240 octets. et on remarque que l'entropie est de 4.889 bits par caractère. Le second terminal correspond à la concaténation de deux textes de la bibliothèque de babel de 3240 octets chacun. On remarque que l'entropie est de 4.891 bits par caractère.

2.2 Question 3

On observe que les entropies des deux textes sont très proches, et toutes deux avoisinent les 5 bits par caractère. Cela indique que les caractères sont distribués de manière quasi aléatoire, avec peu ou pas de redondance exploitable. Le fait que le second texte soit deux fois plus long que le premier, tout en conservant une entropie similaire, confirme que chaque caractère suit une loi pratiquement uniforme. On peut donc conclure que les textes issus de la Librairie de Babel sont générés de façon presque aléatoire.

2.3 Question 4

La bibliothèque de Babel génère des textes en combinant aléatoirement des caractères de son alphabet. Cela signifie que les pages de ses livres ont une entropie très élevée (proche de 5 bits par caractère) et donc une redondance très faible. En conséquence, ces textes sont pratiquement incompressibles car il n'y a pas de motifs répétitifs ou de structures exploitables pour la compression.

3 Histogrammes

Nous avons généré quatres histogrammes

3.1 Question 3

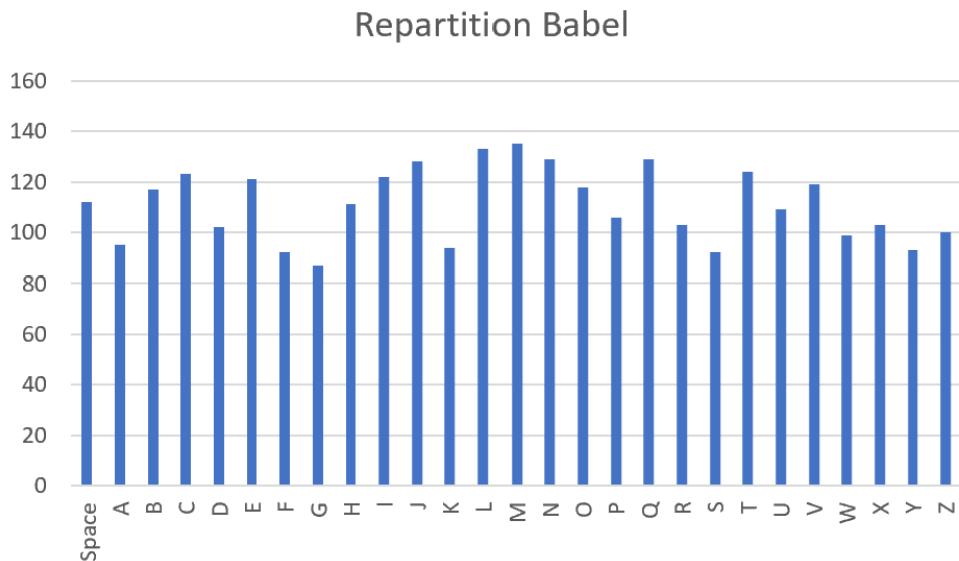


FIGURE 1 – Histogramme des fréquences des lettres dans le texte extrait de la Librairie de Babel.

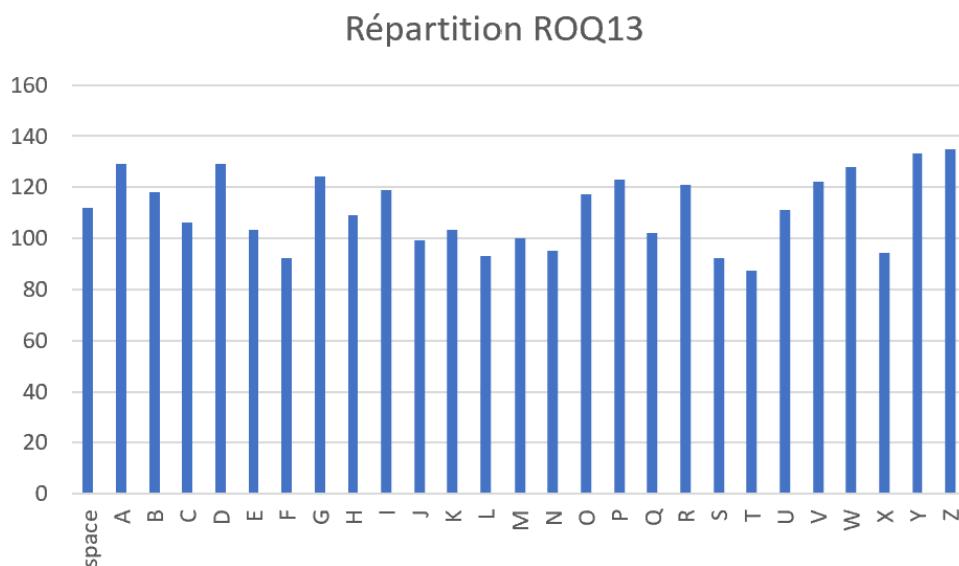


FIGURE 2 – Histogramme des fréquences des lettres après chiffrement ROT13 du texte de Babel.

3.2 Question 5

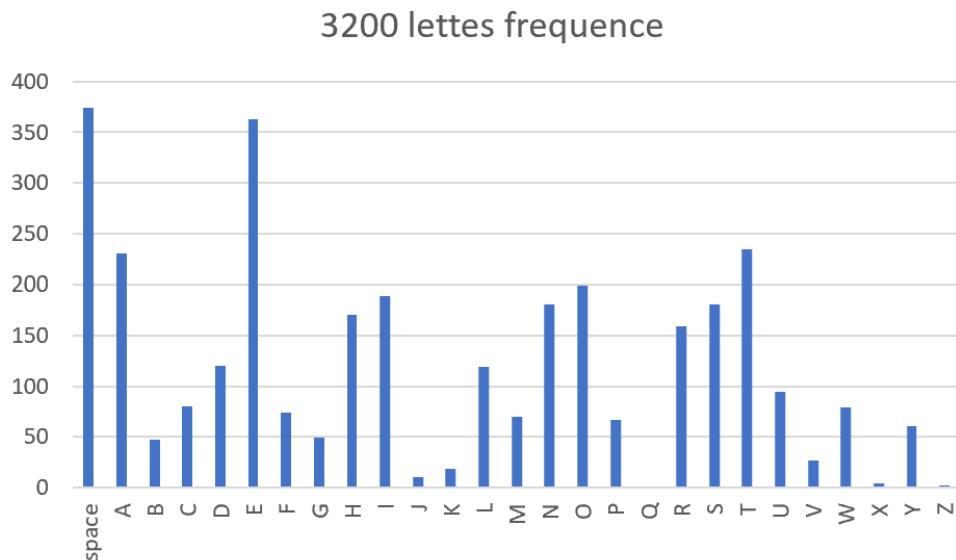


FIGURE 3 – Histogramme des fréquences des lettres dans le texte généré avec `lettre`.

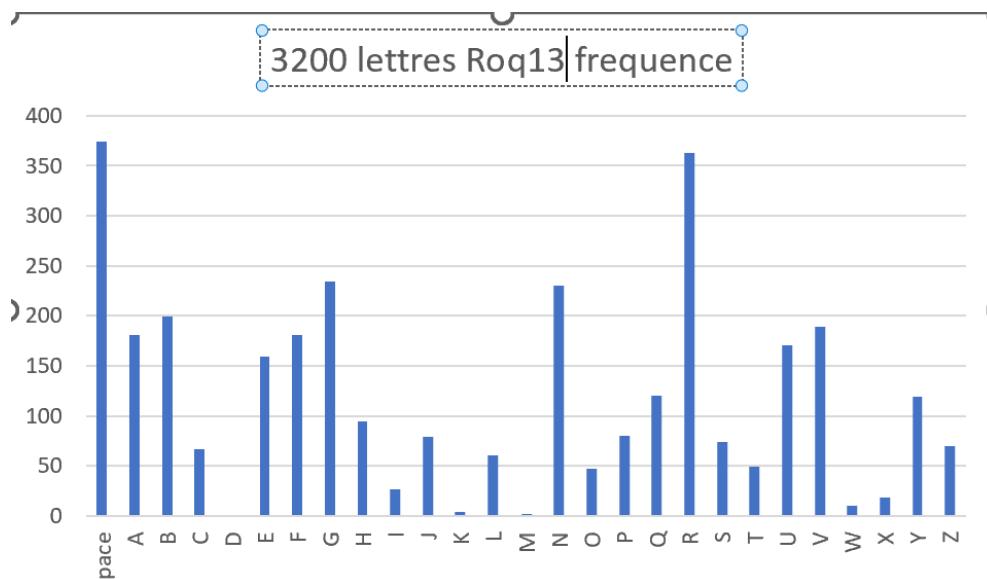


FIGURE 4 – Histogramme des fréquences des lettres après chiffrement ROT13 du texte généré avec `lettre`.

3.3 Question 6

On remarque premièrement que les histogrammes des textes générés avec `babel` sont très plats, indiquant une distribution uniforme des dans Babael. Cependant, on remarque que lettre présente des pics de fréquences pour certaines lettres, suggérant une distribution non uniforme. Les lettres sont proportionnellement plus fréquentes que d'autres, ce qui est typique des textes en langue Anglaise. On remarque que une fois chiffré par ROQ13, les distributions sont plus ou moins les memes, seulement, les pics ne sont plus sur les mêmes lettres. Si on avait fait l'études sur deux lettres, on aurait vu des pics sur les doublons de lettres utilisé souvent en anglais comme "th" ou "in". Cependant sur la distribution de babel, se serait toujours réparti uniformément.

3.4 Question 7

Dans le cas du texte généré par `babel`, la génération étant aléatoire, comptabiliser les fréquences sur deux lettres ne faciliterait pas le déchiffrement. Les motifs seraient répartis de manière uniforme, sans structure exploitable. En revanche, pour le texte généré avec `lettre`, l'analyse des bigrammes permettrait de repérer des séquences typiques de l'anglais. Cela faciliterait l'identification de motifs linguistiques et pourrait aider à reconstruire le sens du message, même après chiffrement.

4 Masque jetable

4.1 Question 1

Voici le code, avec les explications en commentaires :

Listing 1 – Code du chiffrement par masque jetable

```
#Génération d'un masque de taille length avec des bits aléatoires grâce
#à la bibliothèque random
import random
def generate_mask(length=10):
    return [random.randint(0,1) for _ in range(length)]

#Méthode plus sécurisée pour générer des bits aléatoires
#Dans ce cas, le random créé est plus robuste car il se base sur le
#système
# d'exploitation et non pas sur une graine qui pourrait être trouvé. (cas de random)

import secrets

def generate_secure_mask(length=10):
    return [secrets.randbits(1) for _ in range(length)]

#Importation du texte
text = "E ...RK"
texte_sans_espace = text.replace(" ", "")[:100]

# On génère les deux clés de bonne taille.
taille_cle = len(texte_sans_espace) * 8
cle_random = ''.join(str(bit) for bit in generate_mask(taille_cle))
cle_secure = ''.join(str(bit) for bit in generate_secure_mask(taille_cle))
    )

print("clé random", cle_random)
print("clé sûre", cle_secure)
```

4.2 Question 3

Voici les résultats obtenus :

```
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement#
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# echo 2489005-2217813 '1-10-2025'
2489005-2217813 1-10-2025
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ls
cle_random.txt cle_secu.txt frequency h-ascii h-bit h-lettre lettre masque mon_texte.txt stockage texte texte_chiffre_random.txt texte_chiffre_secu.txt
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-bit < mon_texte.txt
0 = 474
1 = 334
Nombre total de bits : 888
Entropie du texte entre : 0.978234
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-bit < texte_chiffre_random.txt
0 = 343
1 = 457
Nombre total de bits : 800
Entropie du texte entre : 0.985302
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-bit < texte_chiffre_secu.txt
0 = 346
1 = 455
Nombre total de bits : 800
Entropie du texte entre : 0.986319
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-ascii < mon_texte.txt
Nombre total d'octets : 101
Entropie de l'entree : 4.534926
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-ascii < texte_chiffre_random.txt
Nombre total d'octets : 100
Entropie de l'entree : 4.518689
(base) root@AntoineP:~/Poly/PolyInte/Cybersecu/TP1/utilitaireTP1/Source - Entropie - Chiffrement# ./h-ascii < texte_chiffre_secu.txt
Nombre total d'octets : 100
Entropie de l'entree : 4.601335
```

FIGURE 5 – Terminal contenant les résultats d'entropie

4.3 Question 4

Nous avons donc obtenus une entropie (calculé par h-bit) avant avoir appliqué le masque de 0.978. Après avoir appliqué le masque généré par la bibliothèque random de base de python, nous avons obtenus une entropie de 0.985 et enfin, après avoir appliqué le masque généré par la bibliothèque "secrets", nous avons obtenus une entropie de 0.986.

De plus, dans le cas du calcul d'entropie par h-ascii, nous avons obtenus une entropie avant application du masque de 4.53 bits/caractère. En appliquant le masque généré par la bibliothèque random, nous avons obtenus une entropie de 4.52 bits/caractère et enfin, en appliquant le masque généré par la bibliothèque "secrets", nous avons obtenus une entropie de 4.60 bits/caractère.

Concernant la robustesse des générateurs de nombre aléatoire, on remarque que le masque généré par la bibliothèque secret est plus robuste car l'entropie calculé est plus basse. Cela signifie que l'aléatoire généré est moins prévisible que celui de la bibliothèque random.

Ces résultats confirment que l'utilisation de `secrets` permet de générer des masques plus aléatoires et donc plus efficaces pour des applications nécessitant une forte entropie. À l'inverse, la bibliothèque `random`, bien qu'adéquate pour des usages généraux, ne garantit pas une entropie suffisante dans des contextes de sécurité.

5 Communication à clé publique, HTTPS et SSL

5.1 Question 1

HTTP envoie les données en clair, sans confidentialité ni vérification du serveur. HTTPS chiffre les échanges et garantit l'identité du site grâce à un certificat SSL/TLS [1].

5.2 Question 2

Le site du dossier étudiant n'est accessible qu'en HTTPS car il manipule des informations sensibles et doit protéger les données des étudiants. Une tentative en HTTP échoue donc, car le serveur refuse les connexions non sécurisées. La solution

sécuritaire est d'installer un certificat valide et de rediriger automatiquement toutes les requêtes HTTP vers HTTPS [2].

5.3 Question 3

Le header Strict-Transport-Security (HSTS) indique au navigateur de n'accepter que des connexions HTTPS pour un site pendant une durée donnée (max-age). Ainsi, même si l'utilisateur tape `http://`, le navigateur force automatiquement le passage en HTTPS. Cela empêche les attaques de type “downgrade” ou interception qui exploiteraient une connexion non chiffrée [3].

5.4 Question 4

Un certificat à clé publique associe une identité (comme le nom de domaine d'un site) à une clé publique. Le navigateur vérifie l'authenticité du certificat en suivant une chaîne de confiance : il contrôle que le certificat du site est signé par une autorité de certification (CA) reconnue et que le nom de domaine correspond, avant d'autoriser la connexion [4].

5.5 Question 5

Les captures ci-dessous présentent le certificat auto-signé généré avec OpenSSL (cert.pem). On peut y voir les informations principales : numéro de série, algorithme de signature, émetteur, sujet, période de validité, ainsi que les extensions comme le Subject Alternative Name (SAN).

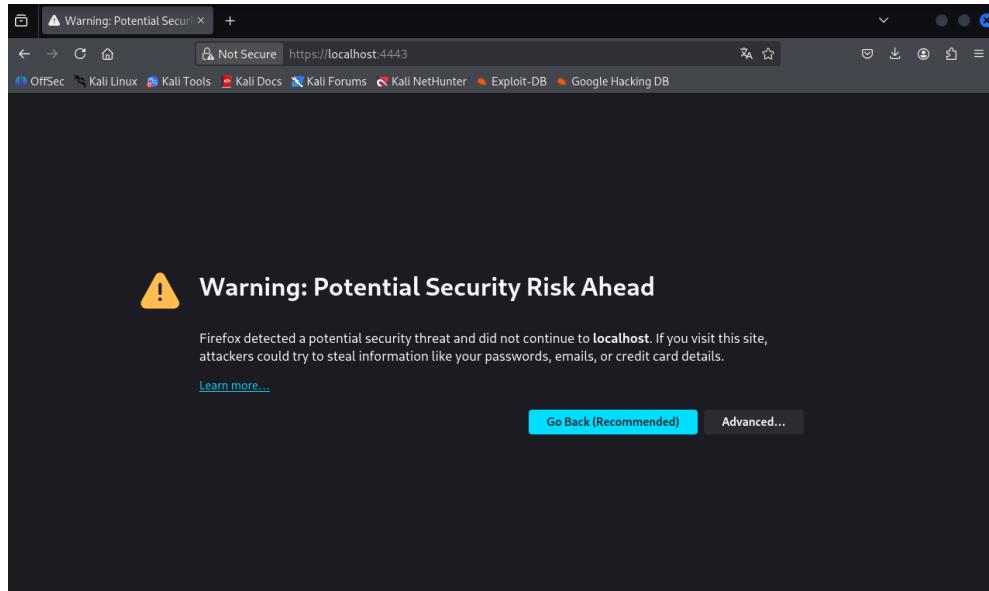
Champ	Valeur
Version	3 (0x2)
Serial Number	52:bc:29:49:6b:b9:5f:6f:db:fc:2a:39:4c:5e:84:7c:af:fc:f8:30
Signature Algorithm	sha256WithRSAEncryption
Issuer	C=CA, ST=QC, L=Montreal, O=TP1, OU=INF8085, CN=localhost Validity Not Before: Sep 30 20:43:55 2025 GMT
Subject Public Key Info	Not After : Sep 30 20:43:55 2026 GMT Public Key Algorithm: rsaEncryption Public-Key: (2048 bit) Modulus: 00:b8:64:cb:72:df:4e:17:7a:61:f6:74:1b:a3:70: 6b:55:ec:79:13:22:7c:0b:08:59:6a:b2:d5:07:1a: df:50:c9:72:da:2b:9f:48:34:3f:cc:50:7f:bf:be: 66:69:ef:21:f5:ce:4e:42:8b:54:d1:21:36:c7:fa: 3c:49:99:f3:5b:46:68:94:40:fd:0c:90:7e:28:4f: 8f:af:60:1e:19:2e:e9:0f:65:a2:28:d5:b3:6c:e7: bc:98:51:0a:a4:3e:2f:64:a3:cf:03:fd:17:a8:e8: 0c:1a:a1:d3:de:8c:ca:54:ca:aa:e7:18:83:0d:a9: b5:bc:ba:e9:1e:8f:1d:30:59:22:89:94:79:95:8d: 01:f8:d3:d3:53:fe:c2:59:2d:6a:f7:7a:99:11:55: 27:f0:0a:c4:ef:12:f3:e5:58:eb:e3:dc:8a:57:0c: 36:88:f8:e6:46:3c:05:61:cc:bf:3e:55:bd:37:13: ab:19:7a:d9:0f:c5:ea:bc:a0:dc:fc:1c:63:ce:c8: d7:7c:a9:cb:b4:8f:65:fe:45:bf:9e:52:36:49:5e: cb:42:5c:25:e0:6f:cb:2e:27:ff:53:f3:d8:97:3a: a1:9a:3a:87:46:4d:08:8c:9b:fc:9d:b0:f6:8e:ff: 38:56:7a:02:fe:41:e1:43:3e:5b:ba:e6:d7:ca:76: 2c:79 Exponent: 65537 (0x10001) X509v3 extensions: X509v3 Subject Key Identifier: 9A:69:82:49:EC:4A:8B:64:F2:4A:A1:06:13:EE:47:24:F7:31:74:08 X509v3 Authority Key Identifier: 9A:69:82:49:EC:4A:8B:64:F2:4A:A1:06:13:EE:47:24:F7:31:74:08 X509v3 Basic Constraints: critical CA:TRUE X509v3 Subject Alternative Name: DNS:localhost, IP Address:127.0.0.1, IP Address:0:0:0:0:0:1
Signature Algorithm	sha256WithRSAEncryption
Signature Value	6b:8e:c9:41:77:87:f7:57:10:91:5d:b8:60:8d:5d:70:14:1e: ab:20:45:c1:26:8e:15:67:2b:35:40:98:b3:c5:99:cb:ba:e0: 08:80:39:e3:2b:36:1f:07:49:fd:88:2c:09:6f:4c:d1:69:8b: 54:72:4d:82:5a:48:ae:e9:39:3b:7b:af:03:f1:b2:fd:22:8e: a5:22:ec:1f:45:86:96:49:57:61:ed:8a:25:f3:6a:d8:8f:e6: 75:17:2a:ce:fd:c1:f8:a1:1c:8f:15:f8:37:79:4b:82:16:73: e9:c7:27:7d:8e:44:48:7a:ca:c9:d0:e4:f0:f6:85:b9:34:4a: 6c:34:d1:18:39:de:e5:7c:9f:11:12:80:db:87:a4:ca:b4:f8: 85:01:91:3e:70:ff:b0:1e:95:b5:78:04:3b:83:12:b7:47:9b: cf:a1:43:7b:38:ea:0e:6e:e4:4d:2c:4b:af:2c:53:d9:55:07: 83:b0:6f:49:20:3d:5f:6b:a0:30:e2:6c:4d:91:82:16:83:93:
	b5:6a:4f:63:56:f2:4f:e6:d3:c5:89:41:a7:42:5a:f2:a7:40: 4c:53:d5:dd:07:11:6e:14:1b:40:f0:31:7b:a2:f1:8c:a2:f9: ca:54:e7:cc:0b:b9:e7:e0:88:e2:5e:4c:0d:d1:c3:51:31:b4: ee:7b:aa:5f

5.6 Question 6

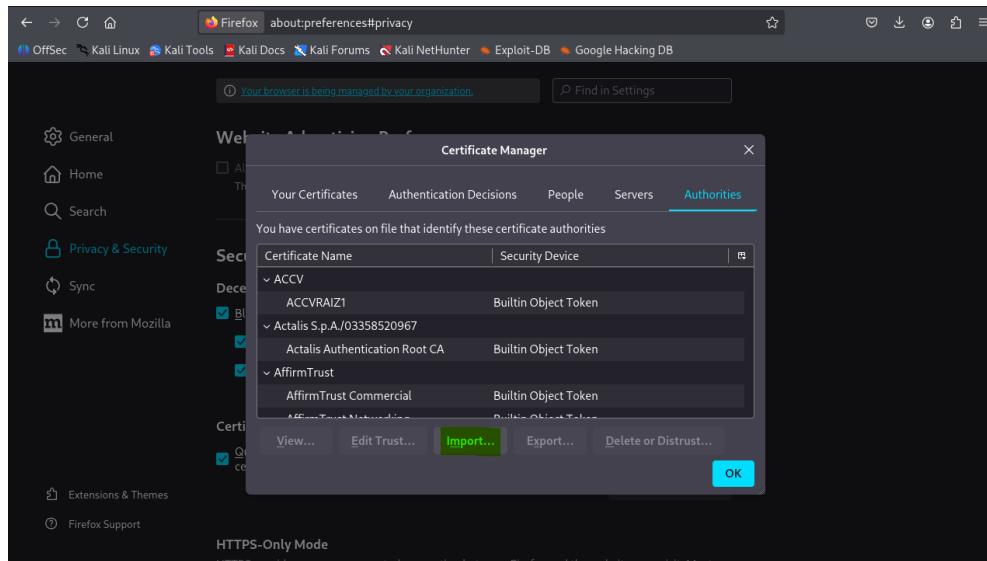
Voici un lien github qui comporte le script python : <https://gist.github.com/SeanPesce/af5f6b7665305b4c45941634ff725b7a>

```
└─(abdelmalek㉿kali)-[~/Documents/INF8085]
$ echo 2217813-2489005 `date`
2217813-2489005 mar 30 sep 2025 17:20:52 EDT
└─(abdelmalek㉿kali)-[~/Documents/INF8085]
$ curl -L -o https_server.py https://gist.github.com/SeanPesce/af5f6b766530b4c45941634ff725b7a/raw/https_server.py
  % Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
     0     0     0      0      0      0      0      0      0      0
  100  1255  100  1255    0      0  --::--  --::--  --::--  4403
└─(abdelmalek㉿kali)-[~/Documents/INF8085]
$ python3 https_server.py 4443 cert.pem private.key
```

5.7 Question 7



Lorsqu'on tente d'accéder au serveur HTTPS local avec Firefox, le navigateur affiche une erreur de sécurité : le certificat utilisé est auto-signé et n'a pas été émis par une autorité de certification (CA) reconnue. Cela signifie que Firefox ne peut pas vérifier l'identité du site et considère donc la connexion comme non sécurisée. Pour contourner ce problème, deux approches existent. La plus simple est d'accepter manuellement le risque dans Firefox en ajoutant une exception de sécurité, ce qui permet d'accéder malgré tout au site. La solution correcte consiste plutôt à créer une autorité de certification locale, signer notre certificat avec celle-ci, puis importer cette autorité dans Firefox. Ainsi, le navigateur reconnaît notre certificat comme valide et n'affiche plus d'erreur.



Dans Firefox, il suffit d'aller dans les paramètres, section Certificates, puis d'importer notre CA locale pour que le certificat soit reconnu et éviter l'erreur.

6 Codage

6.1 Question 1

Premièrement, l'alphabet σ correspond à l'alphabet que peut composer le client pour écrire sont NIP. C'est à dire : "0-9" et "A-Z".

Puis, l'alphabet τ est celui en sortie du codeur. C'est donc un alphabet d'octet qui permet d'encoder du ASCII soit $\{0, 1\}^8$.

Puis, l'alphabet τ' est l'alphabet dans lequel est chiffré le NIP codé. C'est à dire, comme le NIP codé est de 8 octets soit 64 bits, cette alphabet est donc un bloc de 64 bits soit l'alphabet est $\{0, 1\}^{64}$.

6.2 Question 2

Les NIP sont composés de 4 caractères. Donc le langage $L_\sigma = \sigma^4$.

Puis, chaque NIP est répété 2 (est donc écrit en 8 Ascii) fois donc le langage $L_\tau = \{x \in \tau^8 | x = yy, y \in \tau^4\}$.

Enfin, le NIP codé est chiffré par un chiffrement par bloc de 64 bits. Donc le langage $L_{\tau'} = \{x \in \{0, 1\}^{64} | x = encodage(y), y \in L_\tau\}$.

6.3 Question 3

Plusieurs types d'attaques peuvent être envisagés.

Tout d'abord, puisque le NIP est toujours chiffré de la même manière, un attaquant pourrait intercepter un message chiffré et le réutiliser pour tenter une connexion plus tard (attaque par rejet).

Ensuite, la répétition du NIP dans le codage introduit de la redondance. Cette redondance pourrait être exploitée pour analyser le message chiffré et tenter d'en extraire des informations sur le NIP original.

Enfin, bien que l'espace des NIP soit de taille 36^4 , ce qui représente un nombre important de combinaisons, il reste envisageable de mener une attaque par force brute, surtout si l'attaquant dispose de ressources suffisantes.

7 Changement de Codage

7.1 Question 1

Pour chacun des trois codages proposés, voici les attaques du 4.6.3 qu'ils permettent de bloquer :

- **Codage 1** : Le NIP est codé sur 16 bits (14 bits + 2 bits de parité) et concaténé avec un nombre aléatoire de 48 bits. Cela empêche la réutilisation du même message chiffré, car le nombre aléatoire change à chaque fois. Cependant, il n'empêche pas une attaque par force brute sur le NIP, ni l'exploitation d'une éventuelle redondance si le NIP est faible.
- **Codage 2** : Le NIP est codé sur 16 bits, concaténé avec 16 bits aléatoires et un timestamp Unix de 32 bits. Ici, l'ajout du timestamp permet à la banque de rejeter les messages trop anciens, ce qui bloque efficacement les attaques par rejet. Le nombre aléatoire ajoute aussi de l'entropie, rendant plus difficile la recherche de motifs.
- **Codage 3** : Le nouveau et l'ancien NIP sont codés sur 16 bits chacun, puis concaténés avec un timestamp de 32 bits. Ce codage permet de vérifier que le changement de NIP est bien effectué (en comparant l'ancien et le nouveau), et le timestamp bloque les attaques par rejet. Cependant, il n'y a pas de nombre aléatoire, donc si le même changement de NIP est soumis plusieurs fois dans un court laps de temps, le message chiffré sera identique ce qui peut correspondre à une faille dans la sécurité.

7.2 Question 2

Selon moi, le meilleur codage est le **Codage 2**. Il combine un timestamp (pour empêcher les attaques par rejet) et un nombre aléatoire (pour éviter les redondances et rendre chaque message unique, même si le même NIP est soumis plusieurs fois). Cela offre une bonne protection contre les attaques par rejet et rend plus difficile l'analyse statistique ou la force brute sur les messages chiffrés.

8 Chiffrement par bloc et modes d'opération

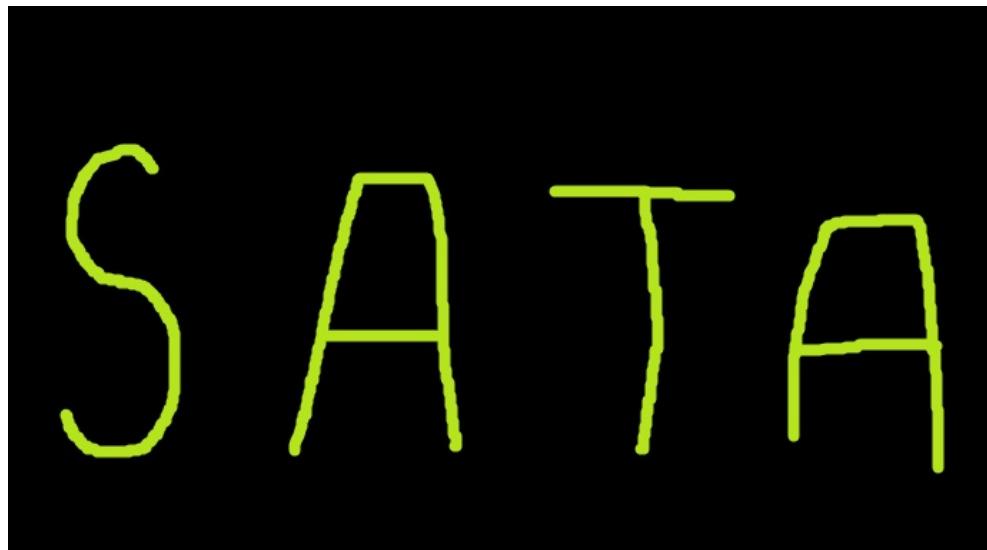


FIGURE 6 – Image originale (mdp.jpg) : mot de passe “SATA” avant chiffrement.

8.1 Question 1

```
└──(venv)─(abdelmalek㉿kali)─[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
  └─$ echo 2217813-2489005 `date`
  2217813-2489005 mer 01 oct 2025 11:59:08 EDT

└──(venv)─(abdelmalek㉿kali)─[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
  └─$ python AES.py -i mdp.jpg -m ECB -o mdp_ecb.jpg
  801570

└──(venv)─(abdelmalek㉿kali)─[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
  └─$ ls -lh mdp_ecb.jpg
  -rw-rw-r-- 1 abdelmalek abdelmalek 123K 1 oct 11:59 mdp_ecb.jpg
```

FIGURE 7 – Exécution du chiffrement en mode ECB avec affichage de la commande et du fichier généré.

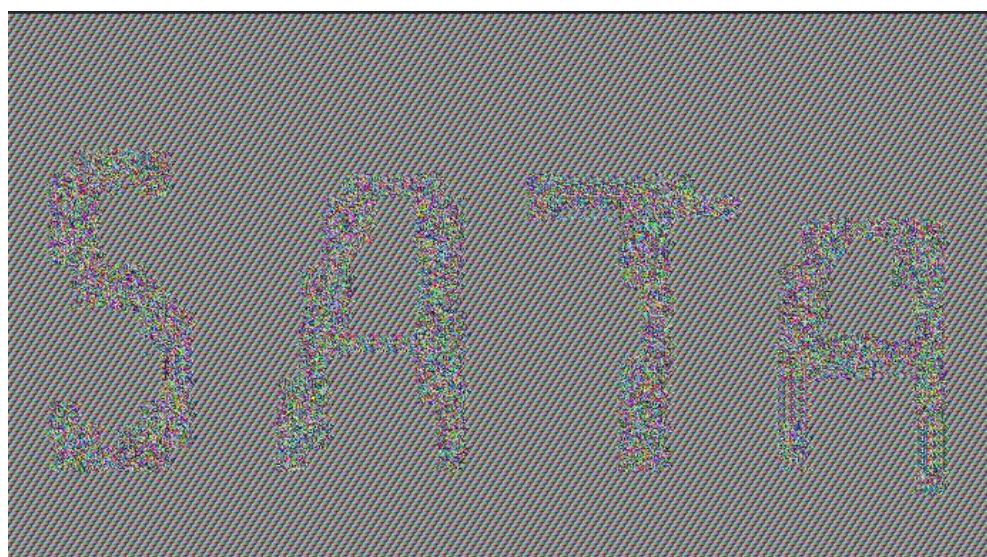


FIGURE 8 – Résultat du chiffrement en mode ECB

En chiffrant l'image avec ECB, on remarque que la structure reste encore visible. On distingue clairement des formes, car des blocs identiques sont chiffrés de la même façon. Cela prouve qu'ECB ne masque pas bien l'information et n'est pas adapté pour protéger des images.

8.2 Question 2

```
[~] abdelmalek@kali:~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC
$ echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 12:05:40 EDT

[~] abdelmalek@kali:~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC
$ python AES.py -i mdp.jpg -m CBC -o mdp_cbc.jpg
301570

[~] abdelmalek@kali:~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC
$ ls -lh mdp_cbc.jpg
-rw-rw-r-- 1 abdelmalek abdelmalek 160K 1 oct 12:05 mdp_cbc.jpg
```

FIGURE 9 – Exécution du chiffrement en mode CBC avec affichage de la commande et du fichier généré.

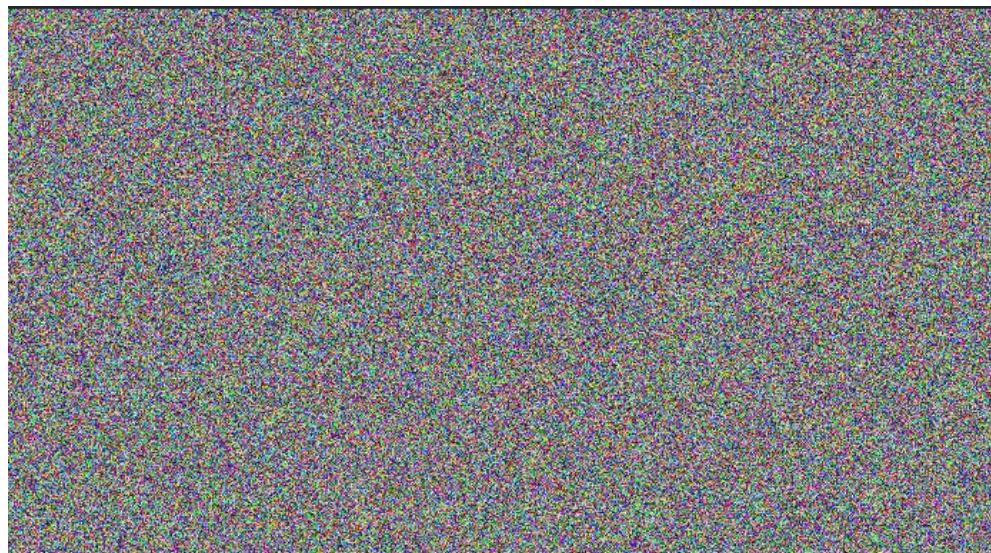


FIGURE 10 – Résultat du chiffrement en mode CBC

Les modes d'opération jouent un rôle très important dans le chiffrement par bloc. Avec ECB, on voit que les blocs identiques donnent un résultat identique, ce qui fait que des motifs apparaissent encore dans l'image et que la confidentialité n'est pas garantie. En utilisant CBC, les blocs dépendent les uns des autres avec un vecteur d'initialisation, ce qui fait disparaître toute structure visible. Cela montre que le choix du mode influence directement le niveau de sécurité et que certains modes comme CBC offrent une bien meilleure protection que ECB.

9 Organisation des mots de passe en UNIX/Linux

9.1 Question 1

```
└─(venv)─(abdelmalek㉿kali)─[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
$ echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 13:33:37 EDT

└─(venv)─(abdelmalek㉿kali)─[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
$ sudo -i
[sudo] Mot de passe de abdelmalek :
└─(root㉿kali)─[~]
# id
uid=0(root) gid=0(root) groupes=0(root)
```

FIGURE 11 – Prompt root et vérification d’identité

```
└─(root㉿kali)─[~]
# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 13:34:58 EDT

└─(root㉿kali)─[~]
# cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:daemon:/usr/sbin/nologin
bin:x:2:bin:/bin:/usr/sbin/nologin
sys:x:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/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:/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
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon:/usr/lib/dhcpcd:/bin/false
mysql:x:101:102:MariaDB Server:/nonexistent:/bin/false
tss:x:102:104:TPM software stack:/var/lib/tpm:/bin/false
strongswan:x:103:65534::/var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:991:991:systemd Time Synchronization:/:/usr/sbin/nologin
_gophish:x:104:106::/var/lib/gophish:/usr/sbin/nologin
iodine:x:105:65534::/run/iodine:/usr/sbin/nologin
messagebus:x:990:990:System Message Bus:/nonexistent:/usr/sbin/nologin
tcpdump:x:106:107::/nonexistent:/usr/sbin/nologin
miredo:x:107:65534::/var/run/miredo:/usr/sbin/nologin
_rpc:x:108:65534::/run/rpcbind:/usr/sbin/nologin
redis:x:109:110::/var/lib/redis:/usr/sbin/nologin
mosquitto:x:110:113::/var/lib/mosquitto:/usr/sbin/nologin
redsocks:x:111:114::/var/run/redsocks:/usr/sbin/nologin
stunnel4:x:989:989:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin
sshd:x:988:65534:sshd user:/run/sshd:/usr/sbin/nologin
dnsmasq:x:999:65534:dnsmasq:/var/lib/misc:/usr/sbin/nologin
Debian-snmp:x:112:115::/var/lib/snmp:/bin/false
sslh:x:113:117::/nonexistent:/usr/sbin/nologin
postgres:x:114:118:PostgreSQL administrator:/var/lib/postgresql:/bin/bash
avahi:x:115:119:Avahi mDNS daemon:/run/avahi-daemon:/usr/sbin/nologin
speech-dispatcher:x:116:29:Speech Dispatcher:/run/speech-dispatcher:/bin/false
_gvm:x:117:120::/var/lib/openvas:/usr/sbin/nologin
usbmux:x:118:46:usbmux daemon:/var/lib/usbmux:/usr/sbin/nologin
cups-pk-helper:x:119:121:user for cups-pk-helper service:/nonexistent:/usr/sbin/nologin
nm-openvpn:x:120:122:NetworkManager OpenVPN:/var/lib/openvpn/chroot:/usr/sbin/nologin
inetsim:x:121:123::/var/lib/inetsim:/usr/sbin/nologin
pipewire:x:986:986:system user for pipewire:/nonexistent:/usr/sbin/nologin
nm-openconnect:x:122:125:NetworkManager OpenConnect plugin:/var/lib/NetworkManager:/usr/sbin/nologin
geoclue:x:123:126::/var/lib/geoclue:/usr/sbin/nologin
lightdm:x:124:127:Light Display Manager:/var/lib/lightdm:/bin/false
lightdm:x:124:127:Light Display Manager:/var/lib/lightdm:/bin/false
statd:x:125:65534::/var/lib/nfs:/usr/sbin/nologin
saned:x:126:128::/var/lib/saned:/usr/sbin/nologin
polkitd:x:985:985:User for polkitd:/:/usr/sbin/nologin
rtkit:x:127:129:RealtimeKit:/proc:/usr/sbin/nologin
colord:x:128:130:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
colord:x:128:130:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
abdelmalek:x:1000:1000:abdelmalek hamadouche,...:/home/abdelmalek:/usr/bin/zsh
```

FIGURE 12 – Affichage de /etc/passwd : contenu complet du fichier /etc/passwd (liste des comptes système et champs associés).

```
└─(root㉿kali)-[~]
  └─# echo 2217813-2489005 `date`
  2217813-2489005 mer 01 oct 2025 13:38:03 EDT

  └─(root㉿kali)-[~]
    └─# awk -F: '{print $1 ":" $2}' /etc/passwd | column -t -s:
root                x
daemon              x
bin                 x
sys                 x
sync                x
games               x
man                 x
lp                  x
mail                x
news               x
uucp               x
proxy               x
www-data             x
backup              x
list                x
irc                 x
_apt                x
nobody              x
systemd-network     x
dhcpcd              x
mysql               x
tss                 x
strongswan           x
systemd-timesync     x
_gophish             x
iodine               x
messagebus           x
tcpdump              x
miredo              x
_rpc                 x
redis                x
mosquitto            x
redsocks             x
stunnel4             x
sshd                x
dnsmasq              x
Debian-snmp          x
sslh                 x
postgres             x
avahi                x
speech-dispatcher    x
_gvm                 x
usbmux               x
cups-pk-helper        x
nm-openvpn            x
inetsim              x
pipewire              x
nm-openconnect         x
geoclue               x
lightdm               x
statd                x

saned                x
polkitd              x
rtkit                x
colord               x
abdelmalek            x
```

FIGURE 13 – Extraction des champs login : password : on voit que la 2^e colonne contient x pour chaque compte (le hash n'est pas ici).

```
└─(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 13:43:15 EDT

└─(root㉿kali)-[~]
└─# ls -l /etc/passwd
-rw-r--r-- 1 root root 3251 30 sep 15:44 /etc/passwd
```

FIGURE 14 – Permissions de /etc/passwd : /etc/passwd est lisible par tous (-rw-r-r-).

```
└─(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 13:44:35 EDT

└─(root㉿kali)-[~]
└─# ls -l /etc/shadow
-rw-r—— 1 root shadow 1429 30 sep 15:42 /etc/shadow
```

FIGURE 15 – Permissions de /etc/shadow : /etc/shadow appartient à root :shadow et a des droits restreints.

J'ai ouvert /etc/passwd et on voit la liste des comptes système. La deuxième colonne contient des x et pas les mots de passe : cela signifie que les hachés des mots de passe ne sont pas stockés dans ce fichier. Les vrais hachés sont dans /etc/shadow, qui est réservé à root (et au groupe shadow). On le voit aussi avec les permissions : /etc/passwd est lisible par tous (-rw-r-r-) parce que des services ont besoin de connaître les comptes, tandis que /etc/shadow a des droits restreints (propriété root :shadow, lecture limitée) pour protéger les hachés contre les accès non autorisés.

9.2 Question 2

```
(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 14:01:51 EDT

(root㉿kali)-[~]
└─# head -n 5 /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

(root㉿kali)-[~]
└─# ls -l /etc/passwd
-rw-r--r-- 1 root root 3251 30 sep 15:44 /etc/passwd

[root@kali)-[~]
└─# ls -l /etc/shadow
-rw-r----- 1 root shadow 1429 30 sep 15:42 /etc/shadow
```

FIGURE 16 – Affichage initial de /etc/passwd et /etc/shadow avec leurs permissions.

```
(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 14:03:43 EDT

(root㉿kali)-[~]
└─# useradd -g users -s /bin/bash -m Abdelmalek

(root㉿kali)-[~]
└─# echo exit_code:$?
exit_code:0
```

FIGURE 17 – Crédation d'un nouvel utilisateur "Abdelmalek" avec useradd (exit code 0 = succès).

```
(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 14:05:00 EDT

(root㉿kali)-[~]
└─# grep '^Abdelmalek:' /etc/passwd || echo "Abdelmalek not in passwd"
Abdelmalek:x:1001:100::/home/Abdelmalek:/bin/bash

(root㉿kali)-[~]
└─# grep '^Abdelmalek:' /etc/shadow || echo "Abdelmalek not in shadow"
Abdelmalek:!::20362:0:99999:7 :::

(root㉿kali)-[~]
└─# ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root 3301 1 oct 14:03 /etc/passwd
-rw-r----- 1 root shadow 1461 1 oct 14:03 /etc/shadow
```

FIGURE 18 – Vérification de l'ajout : nouvelle ligne pour Abdelmalek dans /etc/passwd et entrée correspondante dans /etc/shadow + permissions actuelles de /etc/passwd et /etc/shadow après l'ajout de l'utilisateur.

J'ai utilisé la commande useradd -g users -s /bin/bash -m Abdelmalek pour créer un nouvel utilisateur. Après exécution, on remarque que /etc/passwd contient

une nouvelle ligne pour l'utilisateur Abdelmalek avec son UID, GID, répertoire personnel et shell. Dans la colonne mot de passe, on retrouve simplement x, ce qui signifie que l'information du mot de passe est gérée ailleurs. En parallèle, une nouvelle entrée apparaît aussi dans /etc/shadow avec le champ de mot de passe initialisé à ! ou !!, indiquant qu'aucun mot de passe n'est encore défini pour ce compte. Ainsi, la création d'un utilisateur modifie les deux fichiers : /etc/passwd pour l'identité et les informations générales, et /etc/shadow pour résérer l'espace du mot de passe (qui reste vide tant qu'on ne l'a pas défini). Les permissions des fichiers restent les mêmes : /etc/passwd accessible en lecture par tous, tandis que /etc/shadow est restreint à root et au groupe shadow pour des raisons de sécurité.

9.3 Question 3

```
(root@kali)-[~]
└# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 14:22:09 EDT

└(root@kali)-[~]
└# passwd Abdelmalek

Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès

└(root@kali)-[~]
└# tail -n 5 /etc/passwd
polkitd:x:985:985:User for polkitd::/usr/sbin/nologin
rtkit:x:127:129:RealtimeKit:/proc:/usr/sbin/nologin
colord:x:128:130:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
abdelmalek:x:1000:1000:abdelmalek hamadouche,,,:/home/abdelmalek:/usr/bin/zsh
Abdelmalek:x:1001:100::/home/Abdelmalek:/bin/bash

└(root@kali)-[~]
└# tail -n 5 /etc/shadow
polkitd:*:20361::::::
rtkit:::20361::::::
colord!:20361::::::
abdelmalek:$j9T$ft/hQqEYzm.HlQv9Vc16S/$jtQ449cFLxmf53eobP5wvPQlrByuFQrigauP09EDNkC:20361:0:99999:7:::
Abdelmalek:$y$j9T$mmHwflaerl0.4eJjoB.NA0$6SN1PjGbrfk7l0aVvCoLgn1OacXYg6f6tL6inE1dQC:20362:0:99999:7:::

└(root@kali)-[~]
└# ls -l /etc/shadow /etc/passwd
-rw-r--r-- 1 root root 3301 1 oct 14:03 /etc/passwd
-rw-r----- 1 root shadow 1533 1 oct 14:22 /etc/shadow
```

FIGURE 19 – Exécution de passwd Abdelmalek puis vérification : /etc/passwd et /etc/shadow mis à jour, et permissions des fichiers.

J'ai donné un mot de passe au compte Abdelmalek avec la commande passwd Abdelmalek. Après l'opération, la ligne dans /etc/passwd n'a pas changé pour le champ mot de passe (on y voit toujours x), alors que /etc/shadow a été mise à jour : on observe maintenant une valeur hachée pour Abdelmalek. Cela confirme que l'information du mot de passe n'est pas stockée dans /etc/passwd mais dans /etc/shadow. Les permissions confirment la protection : /etc/passwd est lisible par tous (-rw-r--r--) tandis que /etc/shadow est restreint (root :shadow, lecture limitée), ce qui empêche les utilisateurs non privilégiés d'accéder aux hachés.

9.4 Question 4

```

└─(root㉿kali)-[~]
  # echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 20:04:01 EDT

└─(root㉿kali)-[~]
  # passwd Abdelmalek

Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès

└─(root㉿kali)-[~]
  # tail -n 5 /etc/passwd

polkitd:x:985:985:User for polkitd:::/usr/sbin/nologin
rtkit:x:127:129:RealtimeKit:/proc:/usr/sbin/nologin
colord:x:128:130:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
abdelmalek:x:1000:1000:abdelmalek hamadouche,,,:/home/abdelmalek:/usr/bin/zsh
Abdelmalek:x:1001:100 :: /home/Abdelmalek:/bin/bash

└─(root㉿kali)-[~]
  # tail -n 5 /etc/shadow

polkitd:!*:20361:::::
rtkit:!*:20361:::::
colord:!*:20361:::::
abdelmalek:$y$j9T$ft/hOQEYzm.HlQv9Vc16S/$jtQ449cFLxmf53eobP5wvPQlrByuFQrigauP09EDNkC:20361:0:99999:7:::
Abdelmalek:$y$j9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7:::

└─(root㉿kali)-[~]
  # ls -l /etc/shadow /etc/passwd

-rw-r--r-- 1 root root    3301  1 oct 14:03 /etc/passwd
-rw-r----- 1 root shadow 1533   1 oct 20:04 /etc/shadow

```

FIGURE 20 – Capture : exécution de passwd Abdelmalek et confirmation : montre que le mot de passe a bien été mis à jour (message de succès).

J'ai remis exactement le même mot de passe pour le compte Abdelmalek en relançant passwd Abdelmalek. Après la mise à jour, la ligne dans /etc/passwd n'a pas changé (la colonne mot de passe reste x), alors que la valeur dans /etc/shadow est différente de celle précédente (on voit un nouveau haché).

Avant (4.9.3):\\
Abdelmalek:\$y\$j9T\$mmHwflaerl0.4eJjoB.
NA0\$6SN1PjGbrrfk7l0aVvCoLgn10acXYg6f6tL6inE1dQC:20362:0:99999:7:::\\
Après (4.9.4):\\
Abdelmalek:\$y\$j9T\$KtIOutWbUN6GFyxjfIdwW0\$oGl9p9D.usFJBied4xUv3.0
oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7:::\\

Cela s'explique par le fait que le hachage stocké utilise un salt (et parfois des paramètres comme le nombre de tours) qui est généralement régénéré à chaque modification : même mot de passe → haché stocké différent. L'authentification n'en est pas affectée, car le système utilise le sel stocké dans shadow pour vérifier correctement le mot de passe. Les permissions confirment la protection : shadow est restreint (root :shadow) et passwd reste lisible par tous.

9.5 Question 5

```
(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 20:34:27 EDT

(root㉿kali)-[~]
└─# useradd -g users -s /bin/bash -m Antoine

(root㉿kali)-[~]
└─# tail -n 5 /etc/passwd
rtkit:x:127:129:RealtimeKit:/proc:/sbin/nologin
colord:x:128:130:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
abdelmalek:x:1000:1000:abdelmalek hamadouche,,,:/home/abdelmalek:/usr/bin/zsh
Abdelmalek:x:1001:100::/home/Abdelmalek:/bin/bash
Antoine:x:1002:100::/home/Antoine:/bin/bash

(root㉿kali)-[~]
└─# tail -n 5 /etc/shadow

rtkit:!:20361:::::
colord:!:20361:::::
abdelmalek:$y$9T$ft/HQEYZm.HlQv9Vc16S/$jtQ449cFLxf53eobP5wvPQlrByuFQrigauP09EDNkC:20361:0:99999:7 :::
Abdelmalek:$y$9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7 :::
Antoine:!::20363:0:99999:7 :::

(root㉿kali)-[~]
└─# nano /etc/shadow

(root㉿kali)-[~]
└─# tail -n 5 /etc/shadow

rtkit:!:20361:::::
colord:!:20361:::::
abdelmalek:$y$9T$ft/HQEYZm.HlQv9Vc16S/$jtQ449cFLxf53eobP5wvPQlrByuFQrigauP09EDNkC:20361:0:99999:7 :::
Abdelmalek:$y$9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7 :::
Antoine:$y$9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7 :::
```

FIGURE 21 – Création de l’utilisateur Antoine puis affichage des dernières lignes de /etc/passwd et /etc/shadow (avant édition).

```
(root㉿kali)-[~]
└─# echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 20:46:01 EDT

(root㉿kali)-[~]
└─# grep '^Antoine:' /etc/shadow
grep '^Abdelmalek:' /etc/shadow

Antoine:$y$9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7 :::
Abdelmalek:$y$9T$KtIOutWbUN6GFyxjfIdwW0$oGl9p9D.usFJBied4xUv3.0oz1LZGQ0sd.Qz.3LJ5gD:20363:0:99999:7 :::
```

FIGURE 22 – Édition de /etc/shadow : on copie le haché d’Abdelmalek vers Antoine ; vérification montrant les deux lignes (hachés identiques).

```
(root㉿kali)-[~]
└─# logout

(venv)@(abdelmalek㉿kali)-[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
└─$ echo 2217813-2489005 `date`
2217813-2489005 mer 01 oct 2025 20:52:03 EDT

(venv)@(abdelmalek㉿kali)-[~/.../INF8085/utilitaires_TP1/utilitaireTP1/ChiffrementBLOC]
└─$ su - Antoine
Mot de passe :
(antoine㉿kali)-[~]
└─$
```

FIGURE 23 – Déconnexion de root et tentative de connexion en utilisateur normal : su - Antoine demande le mot de passe et la connexion aboutit.

J’ai créé un deuxième utilisateur (Antoine) puis j’ai ouvert le fichier /etc/shadow. Au départ, son champ mot de passe était vide (!!). J’ai remplacé cette valeur par le haché de l’utilisateur Abdelmalek. Après avoir sauvegardé le fichier et quitté la session root, j’ai essayé de me connecter avec su - Antoine. Le système m’a demandé un mot de passe, et en entrant celui d’Abdelmalek, la connexion a fonctionné. Cela prouve que oui, c’est possible. L’explication est simple : l’authentifi-

cation utilise uniquement le haché présent dans /etc/shadow. Si deux utilisateurs ont exactement le même haché, alors ils partagent le même mot de passe effectif. Le problème, c'est que si quelqu'un arrive à modifier /etc/shadow, il peut donner à n'importe quel compte le mot de passe d'un autre utilisateur, ce qui est une faille de sécurité très sérieuse. C'est pour ça que le fichier /etc/shadow doit être protégé avec des permissions strictes et accessible uniquement par root.

9.6 Question 6

On ne peut pas déchiffrer un haché, parce qu'un hachage est une fonction à sens unique. En effet, on transforme le mot de passe en une valeur fixe et on ne peut pas remonter directement au mot de passe original. Ce que font les attaquants, c'est deviner des mots de passe, les hacher eux-mêmes puis comparer le résultat au haché volé. Les méthodes courantes sont :

- Brute force : essayer toutes les combinaisons possibles (lent si le mot de passe est long).
- Attaque par dictionnaire / règles : tester des mots du dictionnaire et variantes (ajouts de chiffres, majuscules, etc.), c'est beaucoup plus rapide si la victime utilise un mot simple.
- Tables pré-calculées (rainbow tables) : jeux de hachés déjà calculés pour des mots courants (ces tables deviennent inutiles si on utilise un salt).

Et on s'en protège comment ? En utilisant des mots de passe longs et uniques, en stockant les mots de passe avec un salt unique et un algorithme lent/spécialisé, En appliquant des limites de tentatives, et en activant l'authentification multi-facteur. Ces mesures rendent les attaques beaucoup plus difficiles et lentes [5] [6].

10 Choix des mots de passe

10.1 Question 1

```
(abdelmalek㉿kali)-[~/Documents/INF8085]
$ echo 2217813-2489005 "date"
2217813-2489005 jeu 02 oct 2025 20:38:27 EDT

(abdelmalek㉿kali)-[~/Documents/INF8085]
$ ls -lh /usr/share/wordlists/rockyou*
-rw-r--r-- 1 root root 134M 12 mai 2023 /usr/share/wordlists/rockyou.txt
-rw-r--r-- 1 root root 51M 12 mai 2023 /usr/share/wordlists/rockyou.txt.gz

(abdelmalek㉿kali)-[~/Documents/INF8085]
$ john --wordlist=/usr/share/wordlists/rockyou.txt passwords

Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Press 'q' or Ctrl-C to abort, almost any other key for status
123456789      (simple)
sunshine        (brian)
monkey          (action)
liverpool       (vladimir)
4g 0:00:00:01 DONE (2025-10-02 20:39) 3.703g/s 118.5p/s 474.0c/s 474.0C/s 123456 .. diamond
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

(abdelmalek㉿kali)-[~/Documents/INF8085]
$ john --show passwords
vladimir:liverpool:18347:0:99999:7:::
action:monkey:18347:0:99999:7:::
brian:sunshine:18346:0:99999:7:::
simple:123456789:18346:0:99999:7:::

4 password hashes cracked, 0 left
```

FIGURE 24 – Exécution de John the Ripper avec le dictionnaire rockyou.txt sur le fichier passwords : la première commande affiche les mots de passe trouvés (123456789 : simple, sunshine : brian, monkey : action, liverpool : vladimir).

10.2 Question 2

Il ne faut pas utiliser le même mot de passe partout parce que si un service est compromis, l'attaquant pourra ensuite accéder à tous tes autres comptes avec le même mot de passe. Autrement dit, la réutilisation d'un mot de passe multiplie l'impact d'une fuite : une seule fuite suffit pour compromettre plusieurs services. Par exemple, si on utilise le même mot de passe pour une boîte mail et pour un compte universitaire, et que le mot de passe fuit après une attaque contre un site tiers, un attaquant pourra récupérer les e-mails puis demander une réinitialisation de mot de passe sur d'autres services (banque, comptes scolaires, etc.) et prendre le contrôle de ces comptes.

10.3 Question 3

Les deux questions sont liées. Ce qu'on a expliqué en 4.9.6 montre comment un attaquant peut récupérer un mot de passe à partir d'un haché (attaque par dictionnaire, brute force, rainbow tables, etc.). Ce qu'on a dit en 4.10.2 explique pourquoi il ne faut pas réutiliser ce mot de passe. Concrètement, si un attaquant vole un haché sur un site et réussit à le cracker (par exemple avec John + rockyou), il obtient le mot de passe en clair. Si ce même mot de passe est utilisé sur d'autres services, l'attaquant n'a plus qu'à l'essayer partout et peut prendre plusieurs comptes d'un coup.

11 Déchiffrement simple

Pour l'équipe 17 du groupe 3, nous avons eu cette ligne à déchiffrer : LZRHYZRZMN
VFR@YA@ZLSINRRMNRL@RDNEZRLOHSURY@RVSZRZVON@RHJRENLINR@ZLSCVSURLFOVN@RBVZM
HYZRZMNRIHS@NSZRHJRHYFRNUV@WLZYFN@RRMNRL@RLJJNIZNCRZHRFNSCNFRZMNROVWVZL
FTRVSCNENSCNSZRHJRLSCR@YENFVHFRZHRZMNRVXVWR

11.1 Question 1

```
└─(abdelmalek㉿kali)-[~/.../INF8085/utilitaires_TP1/utilitaireTP1/Source - Entropie - Chiffrement]
$ echo 2217813-2489005 `date`
2217813-2489005 lun 06 oct 2025 21:24:49 EDT

└─(abdelmalek㉿kali)-[~/.../INF8085/utilitaires_TP1/utilitaireTP1/Source - Entropie - Chiffrement]
$ ./frequency < ligne17.txt

@ : 6.000 %
A : 0.500 %
B : 0.500 %
C : 3.000 %
D : 0.500 %
E : 2.000 %
F : 4.500 %
H : 5.500 %
I : 2.500 %
J : 2.500 %
L : 6.000 %
M : 4.500 %
N : 11.500 %
O : 2.000 %
R : 18.500 %
S : 6.000 %
T : 0.500 %
U : 1.500 %
V : 6.500 %
W : 2.000 %
X : 0.500 %
Y : 3.500 %
Z : 9.500 %
```

FIGURE 25 – Résultats de l'utilitaire frequency sur le fichier ligne17.txt

11.2 Question 2

Tout d'abord, nous avons décidé d'utiliser un petit script Python pour automatiser le déchiffrement du texte. Ce code permettait de remplacer automatiquement chaque lettre chiffrée par sa correspondance dans la table de substitution que nous avons construite étape par étape. L'utilisation du code n'était pas obligatoire, mais elle nous a permis de gagner du temps et d'éviter les erreurs manuelles pendant le processus de décodage. Voici le code :

```
1 CIPHERTEXT =
2     "LZRHYZRZMNVFR@YA@ZLSINRRMNRL@RDNEZRLOHSURY@RVSZRZVON@RHJRENLINR@ZLSCVSURLFOVN@RB
      VZMHYZRZMNRIHS@NSZRHJRHYFRNUV@WLZYFN@RRMNRL@RLJJNIZNCRZHRFNSCNFRZMNROVWVZLF
      FTRVSCNENSCNSZRHJRLSCR@YENFVHFRZHRZMNRVXVWR"
3 )
4
5 MAPPING = {
6     'A': '', 'B': '', 'C': '', 'D': '', 'E': '', 'F': '', 'G': '', 'H': '', 'I': '',
7     'J': '', 'K': '', 'L': '', 'M': '', 'N': '', 'O': '', 'P': '', 'Q': '', 'R': '',
8     'S': '', 'T': '', 'U': '', 'V': '', 'W': '', 'X': '', 'Y': '', 'Z': '', '@': ''
9 }
10
11 plaintext = ''.join(MAPPING[ch] if MAPPING[ch] else ch for ch in CIPHERTEXT)
12 print(plaintext)
```

Étape 1 - Trouver l'espace ($R \rightarrow \text{espace}$)

J'ai commencé par regarder les fréquences. La lettre R est la plus fréquente (~18.5%). En anglais, l'espace est toujours le symbole le plus présent et il est $\sim 1.07\times$ plus fréquent que la lettre e. J'en ai conclu que R représente l'espace :

```
LZ HYZ ZMNVF @YA@ZLSIN MN ML@ DNEZ LOHSU Y@ VS ZVON@ HJ ENLIN @ZLSCVSU
LFOVN@ BVZMHYZ ZMN IHS@NSZ HJ HYF WNUV@WLZYFN@ MN ML@ LJJNIZNC ZH FNSCNF
ZMN OVWVZLFT VSCNENSCNSZ HJ LSC @YENFVHF ZH ZMN IVXVW
```

Étape 2 - Placer la lettre e ($N \rightarrow e$)

Après l'espace, la lettre la plus fréquente du texte est N (~11.5%), ce qui colle presque exactement avec la fréquence de e (~12.7%). J'ai donc posé $N = e$:

```
Lt HYt tMeVF @YA@tLSIe Me ML@ DeEt LOHSU Y@ VS tV0e@ HJ EeLIe @tLSCVSU
LFOVe@ BVtMHYt tMe IHS@eSt HJ HYF WeUV@WLtYFe@ Me ML@ LJJeIteC tH FeSCeF
tMe OVWVtLFT VSCeEeSCeSt HJ LSC @YEeFVHF tH tMe IVXVW
```

Étape 3 - Placer la lettre t ($Z \rightarrow t$)

La lettre Z (~9.5%) correspond bien à t (~9.1%) et apparaît souvent en début de groupes de lettres, ce qui est typique de mots comme the, to, that. J'ai mis $Z = t$:

```
Lt HYt tMeVF @YA@tLSIe Me ML@ DeEt LOHSU Y@ VS tV0e@ HJ EeLIe @tLSCVSU
LFOVe@ BVtMHYt tMe IHS@eSt HJ HYF WeUV@WLtYFe@ Me ML@ LJJeIteC tH FeSCeF
tMe OVWVtLFT VSCeEeSCeSt HJ LSC @YEeFVHF tH tMe IVXVW
```

Étape 4 - Faire apparaître “the” et “he” ($M \rightarrow h$, $L \rightarrow a$)

On voyait souvent tMe entre espaces. En mettant $M = h$, on obtient the (trigramme le plus fréquent). De plus, Me devient he. Pour préparer has, j'ai aussi posé $L = a$:

```
at HYt theVF @YA@taSIE he ha@ DeEt a0HSU Y@ VS tV0e@ HJ EeaIe @taSCVSU
aFOVe@ BVthHYt the IHS@eSt HJ HYF WeUV@WatYFe@ he ha@ aJJeIteC tH FeSCeF
the OVWVtaFT VSCeEeSCeSt HJ aSC @YEeFVHF tH the IVXVW
```

Étape 5 - “kept” ($D \rightarrow k$, $E \rightarrow p$)

La séquence DeEt correspond naturellement à kept dans l'expression “he has kept ...”. J'ai donc posé $D = k$ et $E = p$:

```
at HYt theVF @YA@taSIE he ha@ kept a0HSU Y@ VS tV0e@ HJ peaIe @taSCVSU
aFOVe@ BVthHYt the IHS@eSt HJ HYF WeUV@WatYFe@ he ha@ aJJeIteC tH FeSCeF
the OVWVtaFT VSCepeSCeSt HJ aSC @YpeFVHF tH the IVXVW
```

Étape 6 - “has” et “us” ($@ \rightarrow s$, $Y \rightarrow u$)

On voyait he ha@ et des blocs Y@. En anglais, has et us sont très probables. J'ai donc mis $@ = s$ et $Y = u$:

```
at Hut theVF suAstaSIE he has kept a0HSU us VS tV0es HJ peaIe staSCVSU
aFOVes BVthHut the IHSsseSt HJ HuF WeUVsWatuFes he has aJJeIteC tH FeSCeF
the OVWVtaFT VSCepeSCeSt HJ aSC supeFVHF tH the IVXVW
```

Étape 7 - “among us” et “times” ($O \rightarrow m$, $U \rightarrow g$)

La séquence aOHSU us devient among us si $O = m$ et $H = o$ (posé plus bas) avec $U = g$. En parallèle, tVmes devient times avec $V = i$ (posé un peu plus loin) :
at Hut theVF suAstaSIE he has kept amHSg us VS tVmes HJ peaIe staSCVsG
aFmVes BVthHut the IHSseSt HJ HuF WegVsWatuFes he has aJJJeIteC tH FeSCeF
the mVWVtaFT VSCepeSCeSt HJ aSC supeFVHF tH the IVXvW

Étape 8 - “their” et “armies” ($V \rightarrow i$, $F \rightarrow r$)

Le motif theVF devient their si $V = i$ et $F = r$. La même paire donne armies ailleurs dans le texte :

at Hut their suAstaSIE he has kept amHSg us iS times HJ peaIe staSCiSg
armies BithHut the IHSseSt HJ Hur WegisWatures he has aJJJeIteC to reSCer
the miWitarT iSCepeSCeSt HJ aSC superiHr tH the IiXiW

Étape 9 - Confirmer “among” et amorcer “standing/consent” ($H \rightarrow o$, $S \rightarrow n$)

Avec $H = o$ et $S = n$, amHSg devient among. Ça oriente aussi stan... (pour standing) et ...onsent (pour consent), qui se confirment après :

at out their suAstanIE he has kept among us in times oJ peaIe stanCing
armies Bithout the consent oJ our WegisWatures he has affecteC to renCer
the miWitarT inCepenCent oJ anC superior to the IiXiW

Étape 10 - “of”, “peace”, “consent”, “affected” ($J \rightarrow f$, $I \rightarrow c$)

Le duo oJ est très probablement of $\rightarrow J = f$. Ensuite peaIe, Ionsent, aJJJeIteC deviennent peace, consent, affected avec $I = c$ (et C fixé ensuite) :

at out their suAstancE he has kept among us in times of peace stanCing
armies Bithout the consent of our WegisWatures he has affecteC to renCer
the miWitarT inCepenCent of anC superior to the ciXiW

Étape 11 - “substance”, “standing”, “render”, “independent”, “and” ($A \rightarrow b$, $C \rightarrow d$)

suAstancE devient substance si $A = b$. Les formes stanCing, renCer, inCepenCent, anC deviennent standing, render, independent, et and avec $C = d$:

at out their substance he has kept among us in times of peace standing
armies Bithout the consent of our WegisWatures he has affected to render
the miWitarT independent of and superior to the ciXiW

Étape 12 - “military” et “legislatures” ($W \rightarrow l$, $T \rightarrow y$)

miWitarT devient military si $W = l$ et $T = y$. Dans le même esprit, WegisWatures devient legislatures :

at out their substance he has kept among us in times of peace standing
armies Bithout the consent of our legislatures he has affected to render
the military independent of and superior to the ciXiL

Étape 13 - “without” et “civil” ($B \rightarrow w$, $X \rightarrow v$)

Bithout devait être without \rightarrow B = w. Enfin **cixil** devient civil avec X = v. On obtient finalement :

at out their substance he has kept among us in times of peace standing armies without the consent of our legislatures he has affected to render the military independent of and superior to the civil

Voici à quoi ressemble la table de substitution finale :

```
MAPPING = {
    'A': 'b', 'B': 'w', 'C': 'd', 'D': 'k', 'E': 'p', 'F': 'r', 'G': '', 'H': 'o',
    'I': 'c',
    'J': 'f', 'K': '', 'L': 'a', 'M': 'h', 'N': 'e', 'O': 'm', 'P': '', 'Q': '',
    'R': '',
    'S': 'n', 'T': 'y', 'U': 'g', 'V': 'i', 'W': 'l', 'X': 'v', 'Y': 'u', 'Z': 't',
    '@': 's'
}
```

Le texte déchiffré : At out their substance he has kept among us in times of peace standing armies without the consent of our legislatures he has affected to render the military independent of and superior to the civil.

Références

- [1] **Transport Layer Security (TLS) - RFC 8446.** <https://datatracker.ietf.org/doc/html/rfc8446> (accès le 30/09/2025).
- [2] **Strict-Transport-Security - MDN Web Docs.** <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security> (accès le 30/09/2025).
- [3] **HTTP Strict Transport Security (HSTS) - RFC 6797.** <https://datatracker.ietf.org/doc/html/rfc6797> (accès le 30/09/2025).
- [4] **What is a secure website certificate? - Mozilla Support.** <https://support.mozilla.org/en-US/kb/secure-website-certificate> (accès le 30/09/2025).
- [5] **NIST SP 800-63B - Digital Identity Guidelines : Authentication and Lifecycle.** <https://pages.nist.gov/800-63-4/sp800-63b.html> (accès le 01/10/2025).
- [6] **Password Storage Cheat Sheet - OWASP Cheat Sheet Series.** https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accès le 01/10/2025).