Epreuve Python

Informations Générales

- ✓ L'épreuve comporte 3 missions à résoudre en **3 heures** consécutives :
  - ***Mission1 :*** *Problème1*
  - ***Mission2 :*** *Problème2*
  - ***Mission3 :*** *Problème3*
- ✓ Chaque équipe doit créer :
  - Un dossier de travail portant le nom **« SalleN°_TableN° »** qui sera utilisé pour enregistrer tous les dossiers et les fichiers solutions de l'épreuve.

    Exemple : Salle1_Table3
  - Trois sous dossiers portant les noms des missions à résoudre **« MissionN°»** qui seront utilisé pour enregistrer tous les fichiers solutions de chaque problème.

    Exemple : Salle1_Table3\Mission1
- ✓ Vous trouvez un dossier à disposition contenant tous les fichiers sources pour chaque mission.

Evaluation & calcul du score

- ✓ Les solutions proposées par les candidats sont évaluées par les membres du jury et notées en fonction de leur exactitude (les bugs ne sont pas tolérés) et de leur vitesse et utilisation mémoire (généralement liées à la complexité en temps et en mémoire de l'algorithme implémenté).
- ✓ Grille de répartition des scores :

Mission1	Mission2	Mission3
30 points	30 points	40 points

Interdiction d'Internet et des outils d'IA

- ✓ Il est strictement interdit d'utiliser Internet, des moteurs de recherche ou tout outil d'intelligence artificielle (ChatGPT, Copilot, Bard, etc.).
- ✓ Toute tentative d'accès à ces outils sera considérée comme une fraude et entraînera l'exclusion de l'examen.

Mission1 [Problème1]

Défi 1 : Investigation sur un trafic réseau : Détection d'Attaque DDoS

Contexte :

Vous êtes un administrateur réseau dans une entreprise de cybersécurité. Un fichier de logs réseau (network_logs.txt) contient des milliers d'entrées de trafic.

Votre mission :

Écrire un programme Python qui analyse ces logs et détecte les adresses IP suspectes générant un trafic anormalement élevé (potentiellement une attaque DDoS).

Exemple d'Exécution :

Entrée (network_logs.txt) : (échantillon de 10 lignes parmi 1000 existants dans le fichier)

①	②	③	④
2023-10-01 12:00:00	192.168.1.18	10.0.0.46	1456367
2023-10-01 12:22:00	192.168.1.21	10.0.0.16	10273578
2023-10-01 13:15:00	192.168.1.49	10.0.0.28	6353234
2023-10-01 14:30:00	192.168.1.12	10.0.0.11	8307427
2023-10-01 15:45:00	192.168.1.41	10.0.0.27	4727928
2023-10-01 16:59:00	192.168.1.47	10.0.0.49	520860
2023-10-01 18:20:00	192.168.1.25	10.0.0.14	1062940
2023-10-01 19:38:00	192.168.1.19	10.0.0.36	297893
2023-10-01 12:30:00	192.168.1.21	10.0.0.18	12000000
2023-10-01 14:35:00	192.168.1.12	10.0.0.15	43204000

① TIMESTAMP
② IP_SOURCE
③ IP_DEST
④ BYTES_TRANSFERRED

Sortie sur ces 10 échantillons (suspect_ips_report.txt) :

```
=== IPs SUSPECTES (Trafic > 20 Mo) ===
Nombre total d'IPs suspectes : 2
 Adresse IP   |     Trafic (Mo)
-----------------------------------------
192.168.1.12  |  49.12 Mo
192.168.1.21  |  21.24 Mo
```

Explication :

- **192.168.1.12** a envoyé 7.92 Mo + 41.20 Mo = 49.12 Mo (Seuil du trafic dépassé >20 Mo)
- **192.168.1.21** a envoyé 9.80 Mo + 11.44 Mo = 21.24 Mo (Seuil du trafic dépassé >20 Mo)

Le reste des Adresses IP seront **Non Suspectes** et ne seront pas affichées dans le rapport " **suspect_ips_report.txt"** (leurs données envoyées cumulées < 20 Mo)

Travail à Faire :

Etape1 : Lire le fichier "network_logs.txt"

- Format : TIMESTAMP IP_SOURCE IP_DEST BYTES_TRANSFERRED
- Exemple : 2023-10-01 12:00:00 192.168.1.1 10.0.0.1 36852744

Etape2 : Analyser le trafic

- Calculer le volume total (en **Mo**) envoyé par chaque IP source .

NB: dans le calcul des cumuls des données transférées par adresse IP, une conversion de l'Octet (dans le fichier source existant) vers le Mo, est exigée et ceci pour simplifier l'affichage.

Etape3 : Détecter les IPs suspectes

- Une IP est suspecte si elle envoie **plus de 20 Mo** après calcul effectué dans l'étape 2.

Etape4 : Générer un rapport (suspect_ips_report.txt)

- Lister les IPs suspectes et leurs nombre avec leur trafic dans un ordre décroissant par leurs données envoyées cumulées en Mo .(Comme expliqué dessus dans l'exemple)
- Afficher dans le rapport, le message **"Aucune activité suspecte détectée"** dans le cas où il n'y aura pas d' Adresses IP suspectes.

Mission2 [Problème2]

Détection d'Intrusions et Contrôle des Permissions

Sous Linux

Dans un système Linux, chaque action effectuée par un utilisateur est enregistrée dans des fichiers de logs (journaux d'activité). De plus, certaines permissions permettent à des utilisateurs d'exécuter des commandes sensibles avec des privilèges (droits d'accès) élevés. Toutefois, des "hackers" peuvent tenter d'exploiter ces permissions pour modifier des fichiers critiques ou exécuter des commandes dangereuses.

Votre mission est de développer un programme Python capable d'analyser ces journaux d'activités et de vérifier les permissions d'administration afin de détecter les risques de sécurité. Le programme devra également générer un rapport contenant des recommandations pour corriger les vulnérabilités identifiées.

Démarche de résolution

1. Collecte et Centralisation des Logs

Vous disposerez de deux fichiers contenant les journaux d'activité (*secure.bin* et *auth.bin*), ayant la structure suivante :

- **timestamp :** Date et heure de l'événement.
- **host** : Nom du serveur.
- **user :** Utilisateur ayant exécuté la commande.
- **command :** Commande exécutée.

Votre première tâche sera de **récupérer** ces données et de les organiser sous forme **d'un tableau d'enregistrement** structurant toutes les informations nécessaires à l'analyse

*Exemple d'échantillon de fichier **"secure.bin"** :*

```
{'timestamp': 'Mar 5 12:01:45', 'host': 'server', 'user': 'user1', 'command': '/bin/ls'}
{'timestamp': 'Mar 5 12:02:50', 'host': 'server', 'user': 'user2', 'command': '/bin/rm -rf /tmp/*'}
{'timestamp': 'Mar 5 12:05:12', 'host': 'server', 'user': 'user3', 'command': '/sbin/reboot'}
{'timestamp': 'Mar 5 12:07:35', 'host': 'server', 'user': 'user4', 'command': '/bin/chmod 777 /etc/passwd'}
{'timestamp': 'Mar 5 12:10:22', 'host': 'server', 'user': 'user1', 'command': '/bin/cat /var/log/auth.log'}
{'timestamp': 'Mar 5 12:15:30', 'host': 'server', 'user': 'user5', 'command': '/usr/bin/python3 /root/script.py'}
{'timestamp': 'Mar 5 12:20:45', 'host': 'server', 'user': 'user6', 'command': '/bin/mkdir /var/test'}
{'timestamp': 'Mar 5 12:25:10', 'host': 'server', 'user': 'user7', 'command': '/bin/chown root:root /etc/shadow'}
{'timestamp': 'Mar 5 12:30:25', 'host': 'server', 'user': 'user8', 'command': '/bin/echo "test" > /root/test.txt'}
{'timestamp': 'Mar 5 12:35:40', 'host': 'server', 'user': 'user9', 'command': '/bin/cp /etc/passwd /tmp/'}
{'timestamp': 'Mar 5 12:41:44', 'host': 'server', 'user': 'user2', 'command': "/bin/echo 'test' > /etc/passwd"}
{'timestamp': 'Mar 5 12:15:20', 'host': 'server', 'user': 'user8', 'command': '/bin/chmod 777 /etc/sudoers'}
{'timestamp': 'Mar 5 12:49:11', 'host': 'server', 'user': 'user5', 'command': '/bin/rm -rf /home/*'}
{'timestamp': 'Mar 5 12:45:51', 'host': 'server', 'user': 'user4', 'command': '/bin/chmod 777 /etc/sudoers'}
{'timestamp': 'Mar 5 12:24:28', 'host': 'server', 'user': 'user3', 'command': '/bin/cp /bin/bash /tmp/shell && chmod +s /tmp/shell'}
{'timestamp': 'Mar 5 12:41:59', 'host': 'server', 'user': 'user6', 'command': '/bin/rm -rf /home/*'}
{'timestamp': 'Mar 5 12:10:47', 'host': 'server', 'user': 'user9', 'command': '/bin/chown root:root /etc/shadow'}
{'timestamp': 'Mar 5 12:57:14', 'host': 'server', 'user': 'user10', 'command': '/bin/chown root:root /etc/shadow'}
{'timestamp': 'Mar 5 12:18:51', 'host': 'server', 'user': 'user5', 'command': '/bin/chmod 777 /etc/sudoers'}
{'timestamp': 'Mar 5 12:10:16', 'host': 'server', 'user': 'user2', 'command': '/bin/cp /bin/bash /tmp/shell && chmod +s /tmp/shell'}
{'timestamp': 'Mar 5 12:31:11', 'host': 'server', 'user': 'user10', 'command': '/bin/ls'}
```

*Exemple d'échantillon de fichier **"auth.bin ":***

```
{'timestamp': 'Jan 07 08:51:00', 'host': 'server2', 'user': 'user1', 'command': '/bin/cat /etc/passwd'}
{'timestamp': 'Jan 02 03:49:00', 'host': 'server2', 'user': 'guest', 'command': "echo 'Hacked' > /root/hacked.txt"}
{'timestamp': 'Jan 05 15:49:00', 'host': 'server2', 'user': 'admin', 'command': '/bin/cat /etc/passwd'}
{'timestamp': 'Jan 07 21:25:00', 'host': 'server3', 'user': 'guest', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 05 17:09:00', 'host': 'server2', 'user': 'user2', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 05 20:21:00', 'host': 'server2', 'user': 'guest', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 07 21:00:00', 'host': 'server1', 'user': 'guest', 'command': '/bin/ls'}
{'timestamp': 'Jan 07 19:18:00', 'host': 'server1', 'user': 'admin', 'command': '/bin/ls'}
{'timestamp': 'Jan 07 14:23:00', 'host': 'server3', 'user': 'guest', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 07 13:47:00', 'host': 'server3', 'user': 'guest', 'command': '/bin/cat /etc/passwd'}
{'timestamp': 'Jan 05 10:37:00', 'host': 'server3', 'user': 'admin', 'command': '/bin/cat /etc/passwd'}
{'timestamp': 'Jan 05 18:09:00', 'host': 'server2', 'user': 'user1', 'command': 'chmod 777 /etc/shadow'}
{'timestamp': 'Jan 01 01:51:00', 'host': 'server2', 'user': 'user1', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 02 08:35:00', 'host': 'server1', 'user': 'guest', 'command': '/sbin/shutdown -h now'}
{'timestamp': 'Jan 06 01:05:00', 'host': 'server1', 'user': 'guest', 'command': '/bin/cat /etc/passwd'}
{'timestamp': 'Jan 04 10:06:00', 'host': 'server3', 'user': 'user2', 'command': '/bin/rm -rf /'}
```

Exemple d'échantillon du tableau :

Mar 5 12:01:45 *Server* *user1* */bin/ls*	*Mar 5 12:02:50* *Server* *user2* */bin/rm -rf* */tmp/**	***.....***	*Mar 5 12:45:51* *server* *user4* */bin/chmod 777* */etc/sudoers*	***.....***	*Jan 01 22:19:00* *server1* *user1* *kill -9 1*	***.....***

2. Génération du Rapport de Sécurité

Le programme devra produire **un rapport récapitulatif dans un fichier texte** intitulé **"Rapport.txt"** contenant le résultat de l'analyse en respectant le format suivant :

- ❖ Un titre "=== Rapport de Sécurité ===".
- ❖ Liste des fichiers sensibles modifiés avec l'action recommandée.
- ❖ Liste des commandes dangereuses exécutées avec l'action recommandée.
- ❖ Les configurations dangereuses dans sudoers avec l'action recommandée.

1. Liste des fichiers sensibles modifiés et actions recommandées

Certains fichiers système sont considérés comme sensibles car ils contiennent des informations essentielles à la sécurité et au bon fonctionnement du système. Toute modification non autorisée de ces fichiers peut représenter un risque majeur.

Fichiers sensibles	Action recommandée
/etc/passwd /etc/shadow /etc/sudoers	Bloquer l'utilisateur et notifier l'administrateur.

Exemple :

{'timestamp': 'Mar 5 12:35:40', 'host': 'server', 'user': 'user9','command': '/bin/cp /etc/passwd /tmp/'}

👉 **Explication** : Le fichier /etc/passwd est un fichier système critique : il contient la liste des utilisateurs du système ainsi que des informations essentielles pour l'authentification, Copier ce fichier peut indiquer une tentative d'accès ou de vol de données sensibles.

👉 **Action recommandée** : Bloquer l'utilisateur et notifier l'administrateur.

2. Liste des commandes dangereuses exécutées

Certaines commandes système peuvent compromettre la sécurité du système si elles sont mal utilisées. L'outil devra identifier l'exécution de ces commandes et proposer des actions correctives.

Commandes dangereuses	Action recommandée
chmod 777 chown root rm -rf shutdown kill -9 1 echo 'Hacked'	Bloquer l'utilisateur et notifier l'administrateur.

Ces commandes peuvent donner un accès non contrôlé à des fichiers sensibles, modifier des permissions critiques ou supprimer des données importantes.

Exemple :

{'timestamp': 'Jan 01 02:30:00', 'host': 'server3', 'user': 'guest', 'command': '/bin/rm -rf /'}

👉 **Explication** : rm -rf / est une commande extrêmement dangereuse. Elle tente de **supprimer tous les fichiers du système**, ce qui entraîne une perte totale des données et rend le serveur inutilisable.

👉 **Action recommandée** : Bloquer l'utilisateur et notifier l'administrateur.

3. Configurations dangereuses dans le fichier sudoers et recommandations de restriction

Le fichier **sudoers** définit les privilèges des utilisateurs pour exécuter des commandes en tant qu'administrateur. Une mauvaise configuration peut permettre à un utilisateur d'exécuter toutes les commandes sans authentification, constituant ainsi une faille de sécurité.

Vous disposerez d'un fichier texte intitulé **"sudoers.txt"**, contenant la liste des privilèges attribués à chaque utilisateur. Un utilisateur est considéré comme **non autorisé** s'il dispose de **tous les privilèges sans exiger de mot de passe** (NOPASSWD: ALL).

Privilège à risque	Action recommandée
NOPASSWD: ALL	Restreindre les privilèges.

Exemple de configuration dangereuse :

user4 NOPASSWD: ALL

👉 **Explication** : Cet utilisateur peut exécuter **toutes** les commandes en tant que root sans authentification. Cela constitue un risque majeur et doit être corrigé.

Extrait de "rapport.txt"

=== Rapport de Sécurité ===

1- Liste des fichiers sensibles modifiés

user1 a exécuté /bin/cat /etc/passwd
admin a exécuté /bin/cat /etc/passwd
guest a exécuté /bin/cat /etc/passwd
admin a exécuté /bin/cat /etc/passwd
user1 a exécuté chmod 777 /etc/shadow
……..
……..

==> action recommandée : Bloquer l'utilisateur et notifier l'administrateur

2- Liste des commandes dangereuses exécutées

guest a exécuté echo 'Hacked' > /root/hacked.txt
user1 a exécuté chmod 777 /etc/shadow
user1 a exécuté /sbin/shutdown -h now
guest a exécuté /sbin/shutdown -h now
user2 a exécuté /bin/rm -rf /
……..
……..

==> action recommandée : Bloquer l'utilisateur et notifier l'administrateur

3- Les configurations dangereuses dans sudoers

user4 NOPASSWD: ALL
==> action recommandée : Restreindre les privilèges.

Mission3 [Problème3]

Chiffrement multi-couche

Vous êtes un expert en cybersécurité engagé par une entreprise internationale pour sécuriser ses communications sensibles. L'entreprise fait face à un certain nombre de menaces provenant de groupes de hackers. Ces menaces incluent des attaques par interception des canaux de communication et des tentatives de déchiffrement des messages critiques.
Les communications traitent de données sensibles, telles que des contrats, des informations financières et des stratégies commerciales. Vous devez concevoir un système de communication sécurisé qui protège ces informations contre les attaques externes. Le système doit être capable de résister aux tentatives de déchiffrement, telles que les attaques par brute force, ainsi qu'aux interceptions et manipulations des messages pendant leur transmission.
Votre mission consiste à concevoir un système de communication sécurisé en mettant en œuvre un principe de chiffrement adapté, à savoir : le chiffrement multi-couches, qui combine plusieurs techniques de protection pour garantir la sécurité des messages :

Étape 1 : Préparation du message à chiffrer et lecture de la clé de cryptage

Avant d'appliquer le chiffrement, le message est transformé pour renforcer sa sécurité. Cette étape comprend plusieurs opérations :

1. **Nettoyage des espaces**
   - Suppression des espaces inutiles en début et fin de message.
   - Conservation d'un seul espace entre les mots.
2. **Remplacement des espaces**
   - Chaque espace restant est remplacé par le symbole **$** pour éviter toute ambiguïté.
3. **Modification de la casse**
   - Conversion des lettres minuscules en majuscules et inversement.
4. **Transformation des chiffres**
   - Chaque chiffre est remplacé par son complément à 9 (exemple : 6 → 3, 5 → 4, 2 → 7, etc.).
5. **Lecture de la clé de chiffrement**
   - La clé de chiffrement est **une séquence numérique de la même longueur** que le message après transformation. Elle sera utilisée dans les étapes suivantes du chiffrement.

Exemple :

En appliquant cette transformation au message :

TunisiA Coding Day V6 By STEM Association, 5 Avril 2025!

Nous obtenons le message chiffré suivant :

tUNISIa$cODING$dAY$v3$bY$stem$aSSOCIATION,4aVRIL$7974!

Exemple de clé de cryptage de taille 56:

99999999998888888888777777777733333333332222222222555555

Étape 2 : Substitution avec S-box

1. **Conversion en binaire du message et de la clé de chiffrement**

- Chaque caractère du message et de la clé est converti en son code ASCII correspondant.
- Ce code ASCII est ensuite transformé en une représentation binaire sur **8 bits**.
- Si la conversion binaire d'un caractère produit moins de **8 bits**, des zéros sont ajoutés à gauche pour compléter la longueur.

Exemples :

Message en binaire :

01110100010101010100111001001001010100110100100101100001001001000110001101001111010
00100010010010100111001000111001001000110010001000001010110010010010001110110001100
11001001000110001001011001001001000111001101110100011001010110110100100100011000010
10100110101001101001111010000110100100101000001010101000100100101001111010011100010
11000010010000110100001001000110000101010110010100100100100101001100001001000011011
10011100100110111001101000100001

Clé de cryptage en binaire :

00111001001110010011100100111001001110010011100100111001001110010011100100111001001
11000001110000011100000111000001110000011100000111000001110000011100000111000001101
11001101110011011100110111001101110011011100110111001101110011011100110111001100110
01100110011001100110011001100110011001100110011001100110011001100110011001100100011
00100011001000110010001100100011001000110010001100100011001000110010001101010011010
10011010010011010100110101001101

2. **Segmentation dynamique des bits**

Une fois le message converti en binaire, il est segmenté en blocs de taille variable selon un processus inspiré du fonctionnement d'un neurone artificiel simple. Voici les étapes suivies pour cette segmentation :

- Tout d'abord, on récupère les 8 premiers bits du message (par exemple : 10011010).
- Ensuite, on convertit ces 8 bits en une valeur décimale : 10011010 en binaire donne 154 en décimal.
- La taille du premier bloc est déterminée en appliquant la formule : 154 mod 7 + 2.

154 mod 7 donne 0, donc on calcule 0 + 2 = 2. Cela signifie que le premier bloc sera de **2 bits**.

- Ensuite, on passe aux 8 bits suivants du message pour déterminer la taille du bloc suivant, et on répète ce processus pour chaque segment du message.
- Le résultat final est un tableau qui contient la taille de chaque bloc, avec une taille dynamique qui varie en fonction des calculs effectués sur chaque segment de 8 bits du message.

Exemple :

Taille_blocs	6	3	3	5	8	5	8	3	3	4	7	5	3	…
	0	*1*	*2*	*3*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	..

3. **Substitution des segments du message avec S-box**

Dans cette étape, nous allons chiffrer le message en remplaçant des blocs de bits par de nouvelles valeurs, à l'aide de tables appelées **S-boxes**. Chaque **S-box** transforme un bloc de bits en un autre bloc, ce qui renforce la sécurité du message.

Nous avons 7 fichiers S-box, chacun correspondant à une taille de bloc différente :

- **2-box.txt** → pour des blocs de 2 bits
- **3-box.txt** → pour des blocs de 3 bits
- **4-box.txt** → pour des blocs de 4 bits
- ...
- **8-box.txt** → pour des blocs de 8 bits

3-box.txt

000 : 101
001 : 110
010 : 011
011 : 100
100 : 010
101 : 001
110 : 111
111 : 000

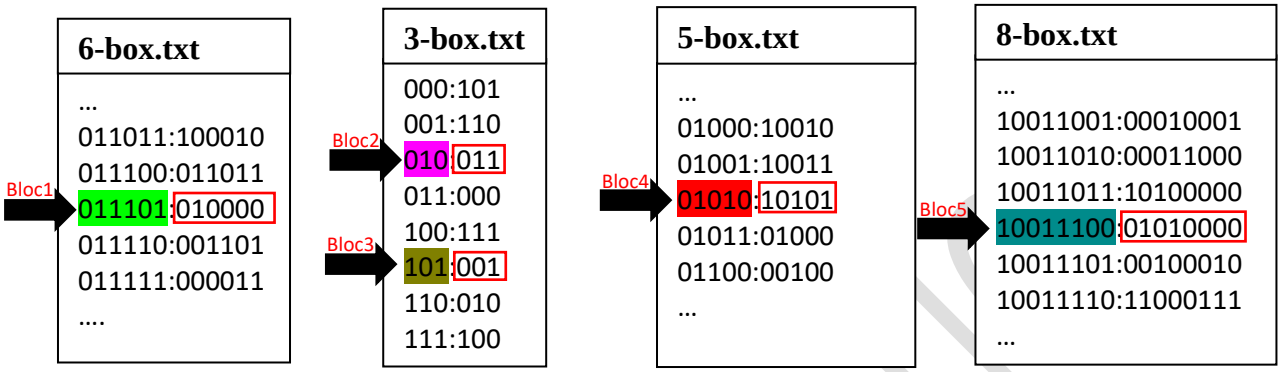
L'idée principale est de parcourir les blocs de bits du message, en utilisant le tableau des tailles de blocs, et d'appliquer une substitution en fonction de la taille de chaque bloc. Cette substitution se fait en accédant au fichier S-box correspondant et en remplaçant chaque bloc binaire par une nouvelle valeur.

Remarque : Si toutes les valeurs du tableau ont été utilisées, le processus reprend depuis le premier élément.

6	3	3	5	8	5	8	3	3	4	7	5	3	…
0	*1*	*2*	*3*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	..

Exemple :

	Bloc1	Bloc2	Bloc3	Bloc4	Bloc5	Bloc6	Bloc7	Bloc8	Bloc9	Bloc10	Bloc11	Bloc12	Bloc13	
Message :	011101	000	101	01010	10011100	10010	01010100	110	100	1001	0110000	10010	010	…
Taille bloc :	6	3	3	5	8	5	8	3	3	4	7	5	3	…



Message résultat : 010000 011 001 10101 01010000 …

Étape 3: Permutation matricielle

La permutation matricielle est une technique qui permet de réorganiser les bits d'un message en appliquant une transformation définie par une matrice de permutation. Cette étape renforce la sécurité en redistribuant les bits des blocs de manière complexe.

1. Remplir une matrice carrée de taille **8×8** où chaque ligne est obtenue en décalant la ligne précédente d'un nombre de positions égal au numéro de la ligne en cours.
   - **Première ligne** : Les indices de 0 à 7
   - **Deuxième ligne** : Décalage circulaire de **1** position vers la droite
   - **Troisième ligne** : Décalage circulaire de **2** positions vers la droite
   - **Quatrième ligne** : Décalage circulaire de **3** positions vers la droite
   - ...
   - **septième ligne** : Décalage circulaire de **7** positions vers la droite

Exemple :

Matrice de permutation :

Ligne 0	0	1	2	3	4	5	6	7
Ligne 1	7	0	1	2	3	4	5	6
Ligne 2	5	6	7	0	1	2	3	4
Ligne 3	2	3	4	5	6	7	0	1
Ligne 4	6	7	0	1	2	3	4	5
Ligne 5	1	2	3	4	5	6	7	0
Ligne 6	3	4	5	6	7	0	1	2
Ligne 7	4	5	6	7	0	1	2	3

2. Application de la Permutation sur le Message

Exemple : 0100000101100110 …

- Diviser le message en blocs de 8 bits.

  `Bloc 1 :` 01000001

  `Bloc 2 :` 01100110

- Appliquer une ligne différente de la matrice de permutation à chaque bloc.
  - `Bloc 1` (ligne 1 → identique) : 01000001
  - `Bloc 2` (ligne 2 → permutation des indices) 01100110
    - `Indice 0 → 7`

- **Indice 1 → 0**
- **Indice 2 → 1**
- **Indice 3 → 2**
- **Indice 4 → 3**
- **Indice 5 → 4**
- **Indice 6 → 5**
- **Indice 7 → 6**
- **Résultat après permutation du bloc2 :** 00110011

- Lorsque toutes les lignes de la matrice sont utilisées, recommencer depuis la première ligne.
- Réassembler les blocs après permutation pour obtenir le message transformé.

Étape 4: Chiffrement avec XOR

Le XOR (OU exclusif) est une opération logique qui compare deux bits :

- Si les bits sont identiques (0 et 0 ou 1 et 1) → le résultat est 0.
- Si les bits sont différents (0 et 1 ou 1 et 0) → le résultat est 1.

Application du XOR au chiffrement :

Appliquer l'opération **XOR** entre chaque bit du message et le bit correspondant de la clé de cryptage (voir étape1)

Exemple :

- **Message après permutation** : 01000010 11010100
- **Clé en binaire** : 00111001 00111001

On applique XOR bit à bit :

```
   01000010
XOR 00111001
= ------------
   01111011
```

```
   11010100
XOR 00111001
= ------------
   11101101
```

Résultat chiffré : 0111101111101101

Cette transformation rend le message illisible sans la clé, assurant ainsi la confidentialité.

Étape 5: Conversion du message chiffré en un format lisible

Pour rendre le message lisible, on le convertit en texte :

- **Diviser** le message binaire en blocs de 8 bits (octets).
- **Convertir** chaque octet en son code ASCII, puis appliquer l'opération **(code ASCII mod 26) + 33** pour obtenir le caractère correspondant.
- **Assembler** les caractères obtenus pour reconstituer le message.

Exemple :

Message chiffré en binaire : 01111011 11101101

- 01111011 → **4** (ASCII 52=123 mod 26 +33)
- 11101101 → **$** (ASCII 36=237 mod 26 +33)

```
4$5*#*7$/5-0*.7%8($:*#'6+(8)##0"7!)3%3"&#!8.13#36764&2(+
```

Annexes sur les fichiers

◉ *Les fichiers textes*

Rôle	En Python
Ouverture d'un fichier avec le mode : ○ "**r**" : Lecture ○ "**w**" : Ecriture (création) ○ "**a**" : Ajout à la fin du fichier	Nom_logique **= open (**'Chemin\Nom_physique' , 'Mode'**)**
Lecture de la totalité d'un fichier	ch = Nom_logique.**read()**
Lecture d'une ligne depuis un fichier texte	ch = Nom_logique.**readline()**
Écriture de la chaine ch dans un fichier texte et retour à une nouvelle ligne	Nom_logique.**write(**ch **+ "\n")**
Retourne **Vrai** si le pointeur est à la fin du fichier sinon elle retourne **Faux** ***N.B.*** : **La fin d'un fichier texte est la chaine vide**	ligne= Nom_logique.**readline()** **while** ligne != "" : Traitement ligne = Nom_logique.readline()
Fermeture d'un fichier	Nom_logique.**close ()**

◉ *Les fichiers de données (binaires/typés)*

Rôle	En Python
Ouverture d'un fichier avec le mode : ○ "**rb**" : Lecture ○ "**wb**" : Ecriture (création) ○ "**ab**" : Ajout à la fin du fichier	Nom_logique **= open (**'Chemin\Nom_physique' , 'Mode'**)**
Lecture d'un objet à partir d'un fichier	from **pickle** import **load, dump** Objet **= load (**Nom_logique**)**
Écriture d'un objet dans un fichier	from **pickle** import **load**, **dump** **dump (**Objet , Nom_logique**)**
Retourne **Vrai** si le pointeur est à la fin du fichier sinon elle retourne **Faux** ***N.B.*** : **La fin d'un fichier binaire est une erreur**	Fin = False **while** not (Fin) : **try :** x = **load** (Nom_logique) **except :** Fin = True
Fermeture d'un fichier	Nom_logique.**close ()**