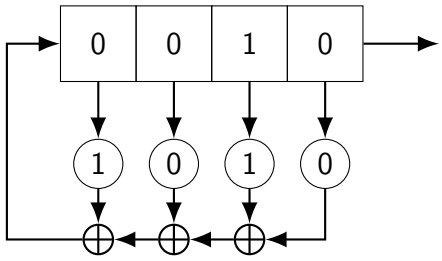
TP 2 : Chiffrements symétriques

Exercice 1 (OTP).

1. Codez une fonction qui permet de chiffrer avec le chiffrement parfait ($C = M \oplus K$). Cette fonction prend en entrée une clé et un message, tous deux sous forme de chaîne de caractères, et sa sortie est également une chaîne de caractères. Vous aurez peut-être l'usage de l'opérateur $\wedge$.
2. Codez la fonction de déchiffrement correspondante.

Exercice 2 (LSFR *).
Soit le LSFR avec la graine $s^{(0)} = [0, 0, 1, 0]$ et les coefficients $c = [1, 0, 1, 0]$.



- Calculez les 10 premiers bits de sortie

Exercice 3 (Openssl et Shell).
L'objectif de cet exercice est de chiffrer une image en ligne de commande (pas en Python, donc) à l'aide des outils fournis par la librairie openssl.

1. Prenez une image de votre choix et convertissez la dans le format `ppm` à l'aide de la commande `convert`.
2. Avec la commande `head`, mettez les 3 premières lignes de votre fichier `ppm` dans un fichier `header.txt`.
3. Avec la commande `tail` mettez tout le fichier `ppm` sauf les 3 premières lignes dans un fichier `body.bin`.
4. Chiffrez avec `openssl` en AES-ECB le fichier `body.bin`.
5. Remettez l'en-tête `header.txt` au début du fichier obtenu à la question précédente pour avoir une image chiffrée, et observez le résultat.

Exercice 4 (Modes ECB et CBC).
Dans cet exercice, vous allez utiliser la bibliothèque `pycryptodome` (exécutez si nécessaire `pip install pycryptodome` dans un terminal) et ses modes de chiffrement AES prédéfinis (voir https://pycryptodome.readthedocs.io/en/latest/index.html).

- Votre script pourra commencer par la ligne `from Crypto.Cipher import AES`.

- Utilisez la clé `YELLOW SUBMARINE`, qui a le bon goût d'être composée de 16 caractères (= 16 octets), comme requis par AES.
- Pour les trois premières questions, vous supposerez que la longueur du message clair est un multiple de 16 octets. Si ce n'est pas le cas, vous couperez les derniers caractères du message clair.
- Choisissez un message long, comportant plus de 16 caractères. Pour éviter les soucis d'encodage, limitez-vous aux caractères ASCII 7 bits (majuscules, minuscules, espaces, ponctuation, pas d'accents, etc.).
- La commande AES prend des `bytes` en entrée. Vous pouvez convertir un `str` en `bytes` en le faisant précéder d'un `b`.
- Pour la lisibilité de l'affichage des chiffrés, vous voudrez peut-être les convertir en hexadécimal en utilisant `.hex()`.

1. Réalisez le mode de chiffrement AES-EBC et le déchiffrement.
2. Réalisez le mode de chiffrement AES-CBC et le déchiffrement. Attention à l'erreur `decrypt() cannot be called after encrypt()`.
3. Réalisez à la main (à partir du mode ECB) le mode de chiffrement AES-CBC et le déchiffrement. Quelques conseils :
   - Découpez le message clair en blocs de 16 caractères.
   - Attention au type d'une lettre d'un bloc (chez moi c'est un `int`).
   - Créez un vecteur initial aléatoire à l'aide des deux lignes `from Crypto.Random import get_ random_ bytes` et `iv= get_ random_ bytes(16)`.
   - Si `a` est un `int`, l'expression `int.to_ bytes(a,1,'big')` produit un `byte`.
4. Si vous avez le temps, réalisez la fonction de padding PKCS 7 et la fonction qui ôte le padding. Le remplissage se fait par octets entiers. La valeur de chaque octet ajouté est le nombre d'octets qui sont ajoutés, c'est-à-dire que N octets, chacun de valeur N, sont ajoutés. Le nombre d'octets ajoutés dépend de la limite de bloc à laquelle le message doit être étendu (pour AES, c'est 16).

   Autrement dit, le remplissage sera l'un des suivants :

   ```
   01
   02 02
   03 03 03
   04 04 04 04
   05 05 05 05 05
   06 06 06 06 06 06
   etc.
   ```

   https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS#5_and_PKCS#7