SAINT LOUIS	**Chap 6** **Chiffrement à clés symétriques**	**E. Le Gars**
B3-S2	TD d'exploration sur Labtainer - Chiffrement avec AES	**BTS1-S2**

Objectifs :

- Comprendre le principe de chiffrement / déchiffrement symétrique
- Effectuer des chiffrements de fichiers avec un algorithme de type AES
- Explorer les différentes méthodes de chiffrement ECB, CBC et CFB

Prérequis : Vidéo Introduction au chiffrement symétrique

TABLE DES MATIÈRES

Contexte :
Votre entreprise **Cibeco** est une pépinière spécialisée dans l'accompagnement de startups travaillant dans le domaine de la transition énergétique. Vous fournissez notamment des locaux et des prestations informatiques (hébergement de serveurs).

Un de vos clients, **Ecotri**, une startup spécialisée dans la valorisation des déchets, fait héberger son application Web dans votre ferme de serveurs ainsi qu'un espace de stockage.

Votre mission va consister à accompagner le déploiement de la solution d'Ecotri au sein de l'infrastructure de Cibeco, en prenant en compte les exigences suivantes :

- ➢ **Disponibilité du service → Redondance des élément de service et d'interconnexion**
- ➢ **Protection des données → Pendant leur acheminement et lors du stockage**

Consignes :
Les éléments **marqués en vert** sont des commandes à rentrer sur les systèmes de la manip
Les éléments **marqués en bleu** sont relatifs à des questions auxquelles vous devez répondre dans les encadrés prévus à cet effet

1. Principes du chiffrement symétrique

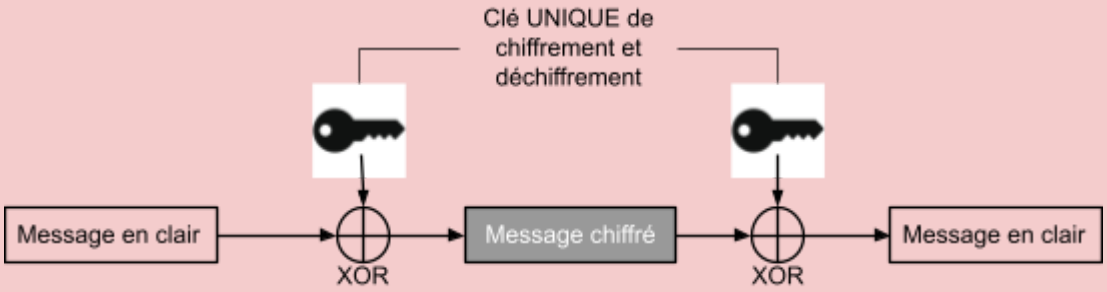
1.1 Exploitation des propriétés du “OU exclusif” (XOR) pour le chiffrement

Le **chiffrement** (et non cryptage) **XOR** est un système de chiffrement basique qui a beaucoup été utilisé dans les débuts de l'informatique et continue à l'être encore aujourd'hui car il est facile à implémenter, dans toutes sortes de programmes.

A	B	(A XOR B)
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	FAUX

Le **XOR** est un opérateur logique qui correspond à un **OU exclusif** : c'est le (A + B) qu'on utilise en logique booléenne mais qui exclut le cas où A et B sont simultanément vrais. On note cette opération (A ⊕ B).

Ce procédé de chiffrement/déchiffrement **XOR** est complètement **symétrique** : la même opération est ré-appliquée au message chiffré pour retrouver le message initial.



Exemple : Voici le mot “MESSAGE” converti en binaire :

Lettres	**M**	**E**	**S**	**S**	**A**	**G**	**E**
ASCII	77	69	83	83	65	71	69
Binaire	01001101	01000101	01010011	01010011	01000001	01000111	01000101

Le mot “**CLE**” en binaire est lui représenté par 01000011 - 01001100 - 01000101 .

Message en clair (binaire)	01001101	01000101	01010011	01010011	01000001	01000111	01000101
Clé en binaire (répétée si nécessaire)	01000011	01001100	01000101	01000011	01001100	01000101	01000011
Message chiffré	**00001110**	**00001001**	**00010110**	**00010000**	**00001101**	**00000010**	**00000110**

Retrouver le message en clair à partir du message chiffré :

Clé en binaire	01000011	01001100	01000101	01000011	01001100	01000101	01000011
Message chiffré (binaire)	**00001110**	**00001001**	**00010110**	**00010000**	**00001101**	**00000010**	**00000110**
Message en clair (binaire)							

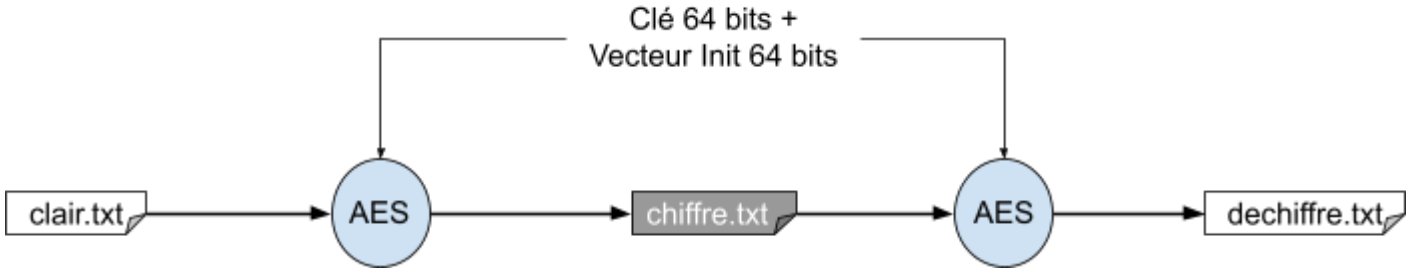
Avantage de ce mode de chiffrement :

Rapidité et économie de traitement en chiffrement et en déchiffrement, avec un opérateur booléen de base demandant peu de ressources

1.2 Chiffrement et Déchiffrement symétrique AES sur un fichier (Labtainer)

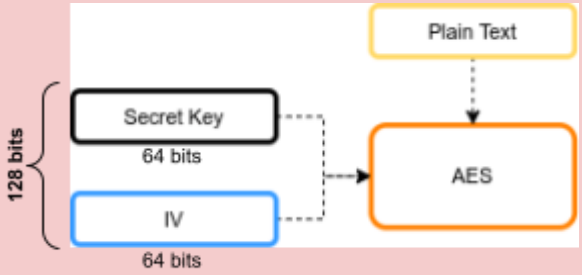
Les algorithmes de chiffrement symétriques sont nombreux et ont des caractéristiques qui justifient leur emploi dans différentes situations : Robustesse au crackage, lourdeur de traitement, lourdeur de stockage, etc. On peut citer comme algorithmes courants : **AES, DES, 3DES, Blowfish, Serpent**

Dans cette partie, nous allons chiffrer et déchiffrer le contenu d'un fichier texte en utilisant la même clé secrète, en utilisant un algorithme de chiffrement standard **AES (Advanced Encryption Standard)**



- ➢ Lancer le labtainer sur le chiffrement symétrique : **labtainer symkeylab**
- ➢ Dans la session "Symkeylab", créer un fichier contenant un texte simple :
  - **$ touch clair.txt** → Création du fichier comportant le message en clair
  - **$ echo "Ceci est un test hu hu hu" > clair.txt** → redirection du texte dans le fichier
  - **$ more clair.txt** → vérification du contenu du fichier
  - **$ touch chiffre.txt** → Création du fichier (vide) comportant le message chiffré

L'outil **OpenSSL** est utilisé pour gérer des communications chiffrées de couche Transport (TLS vu auparavant) et il dispose notamment d'une bibliothèque (Library) d'algorithmes de chiffrement, notamment l'**AES** avec plusieurs modes que nous verrons en détail.



Avec le mode de chiffrement que nous allons tester ici, la clé de 64 bits va être combinée avec un **vecteur d'initialisation (IV, Initialization Vector)** afin de rendre le chiffrement plus robuste. Dans le cadre d'échanges confidentiels, seule la clé doit rester secrète, l'IV étant transmis en clair avec le message chiffré.

L'IV doit faire la même taille que la clé (64 bits)

- ➢ **$ openssl aes-128-cbc -e -in clair.txt -out chiffre.txt -K <clé> -iv <vecteur-d-initialisation>**
  - aes-128-cbc : type d'algorithme (AES avec clé de 64 bits) et de mode de chiffrement (ici cbc)
  - -e : Encrypt
  - <Clé> : la clé doit être en hexadécimal (64 bits)
  - <Vecteur d'initialisation> : en hexadécimal (64 bits)
  - clair.txt → Fichier source (texte en clair)
  - chiffre.txt → Fichier cible (résultat du chiffrement, texte chiffré)

- ➢ La clé et le vecteur d'initialisation comportent chacun 64 bits. Combien de caractères hexadécimaux faut-il pour les représenter ?

- ➢ Lister le contenu du fichier chiffre.txt (utiliser les outils **cat**, **more**, **less**, ou **leafpad**) et commenter le résultat de la tentative d'affichage par un éditeur de texte classique

➢ Analyser le contenu de ce fichier avec l'outil HexDump : **$ hexdump - C chiffre.txt** et essayer d'expliquer pourquoi certains caractères ASCII sont correctement affichés et d'autres pas

➢ Déchiffrer le fichier chiffre.txt avec la clé unique et l'IV :
$ openssl aes-128-cbc -d -in chiffre.txt -out dechiffre.txt -K <clé> -iv <vecteur-d-initialisation>

- -d : Decrypt
- <Clé> : la clé doit être en hexadécimal (128 bits) → **XXXX** caractères hexa
- <Vecteur d'initialisation> : en hexadécimal (128 bits) → **XXXX** caractères hexa
- chiffre.txt → Fichier source (résultat du chiffrement, texte chiffré)
- dechiffre.txt → Fichier destination (texte déchiffré, normalement en clair)

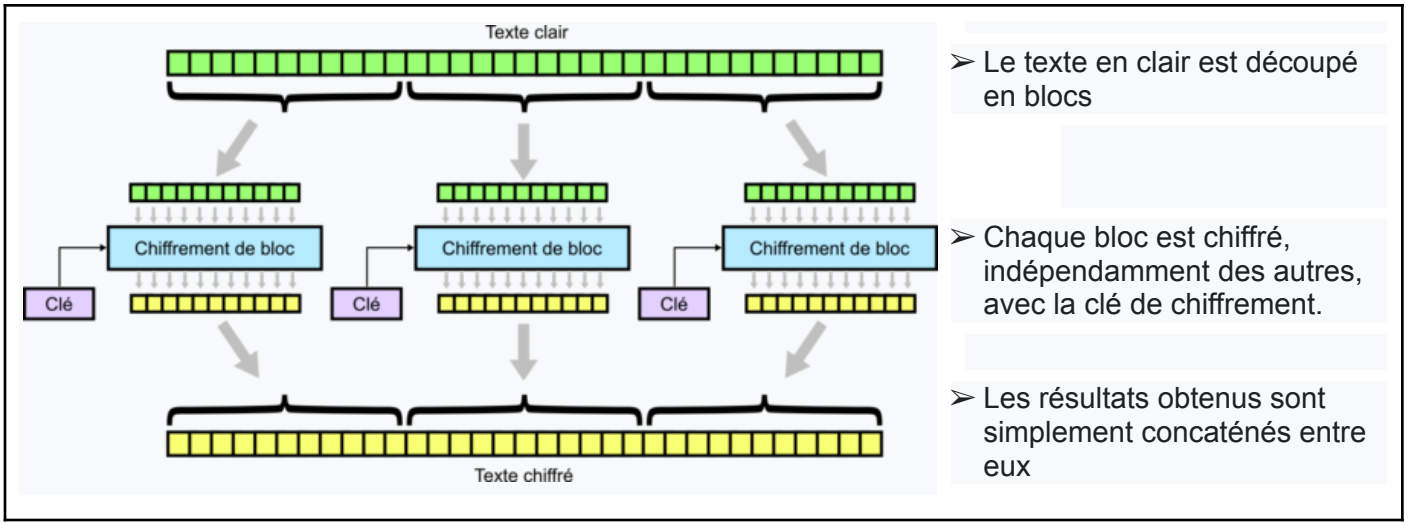
➢ Effectuer une comparaison entre le fichier clair.txt et le fichier dechiffre.txt :
$ diff -a clair.txt dechiffre.txt

2. Comparaison des modes de Chiffrement

Les standards de chiffrement **AES** dispose de plusieurs modes de chiffrement que nous allons tenter de présenter et de comparer entre eux, notamment du point de vue de leur robustesse et de leur lourdeur de traitement.

2.1 Mode “Electronic codebook” (ECB)

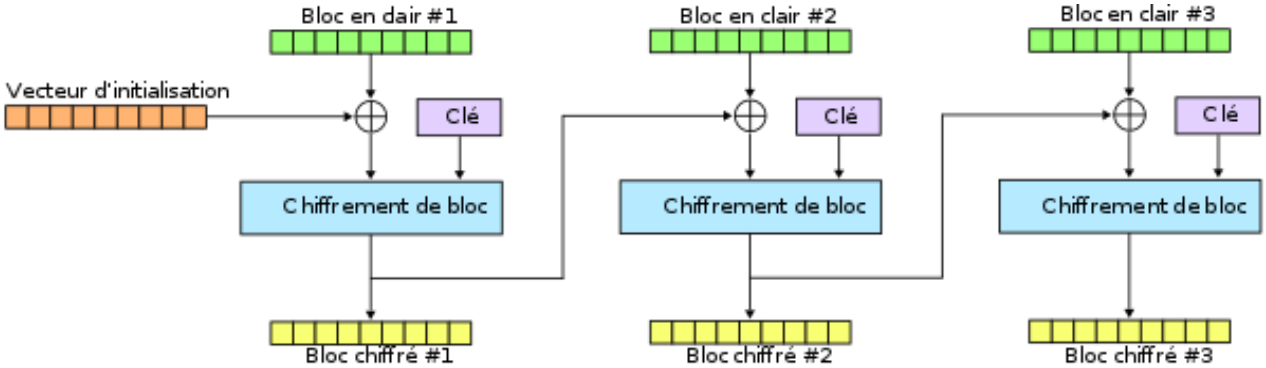
Il s'agit du mode le plus simple. **Le message à chiffrer est subdivisé en plusieurs blocs qui sont chiffrés séparément les uns après les autres.**



Inconvénients et limitations : Le gros défaut de cette méthode est que deux blocs avec le même contenu seront chiffrés de la même manière, on peut donc tirer des informations à partir du texte chiffré en cherchant les séquences identiques. Source : Wikipedia

2.2 Enchaînement des blocs : “Cipher Block Chaining” (CBC)

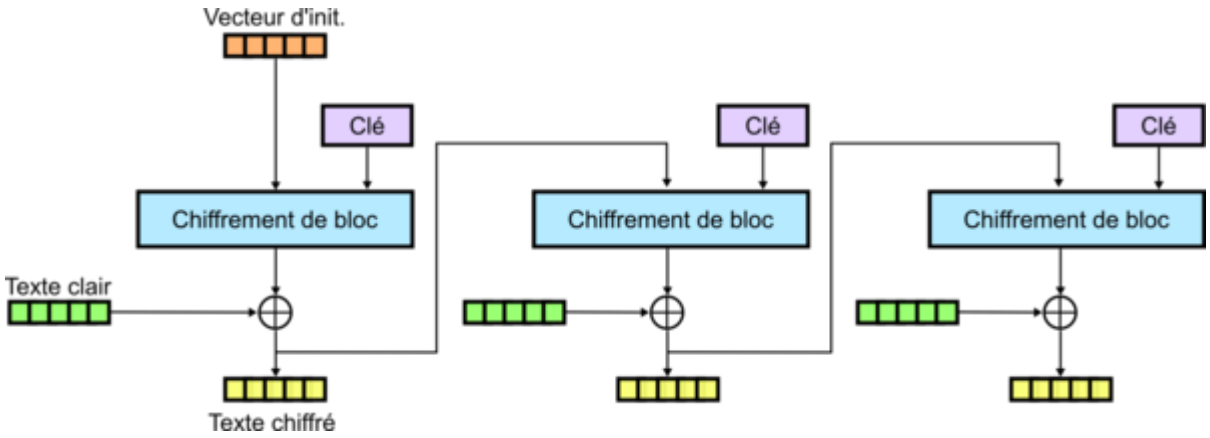
Dans ce mode, on applique sur chaque bloc un ‘OU exclusif’ avec le chiffrement du bloc précédent avant qu’il soit lui-même chiffré. De plus, afin de rendre chaque message unique, un vecteur d’initialisation (IV) est utilisé.



Défauts de CBC : Un des points négatifs de CBC étant qu'il ne peut pas être parallélisé étant donné que le bloc courant nécessite que le précédent soit chiffré. Il est donc séquentiel. En utilisant les mathématiques modernes, il est possible de retrouver l'ensemble du message en clair. Source : Wikipedia

2.3 Chiffrement à rétroaction : “Cipher Feedback” (CFB)

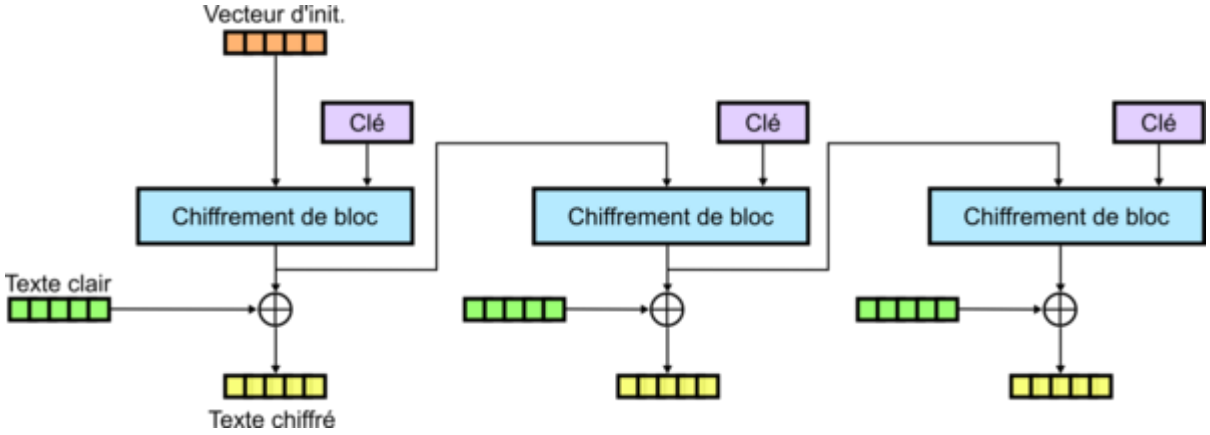
Ce mode et les suivants agissent comme un chiffrement par flux. Ils génèrent un flux de clés qui est ensuite appliqué au document original.



Dans ce mode, le flux de clé est obtenu en chiffrant le précédent bloc chiffré. CFB est un chiffrement par flot. Son grand intérêt est qu'il ne nécessite que la fonction de chiffrement, ce qui le rend moins cher à câbler ou programmer pour les algorithmes ayant une fonction de chiffrement différente de la fonction de déchiffrement (AES, DES, Blowfish ou autre). Source : Wikipedia

2.4 Chiffrement à rétroaction de sortie : “Output Feedback” (OFB)

Dans ce mode, le flux de clé est obtenu en chiffrant le précédent flux de clé.



C'est un mode de chiffrement de flot qui possède les mêmes avantages que CFB. De plus, il est possible de le pré-calculer en chiffrant successivement le vecteur d'initialisation.Source : Wikipedia

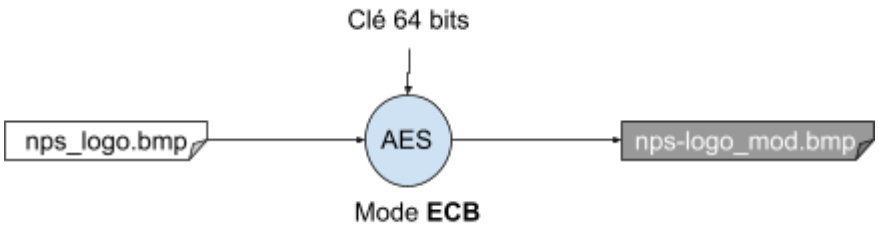
2.5 Éprouver la robustesse des modes de chiffrement sur un fichier d'image BMP

Dans ce labo, un serveur Web tourne avec une page "**index.html**" pointant vers un fichier d'image (logo) "**nps-logo.bmp**" présent dans le répertoire courant de la VM (/home/ubuntu), ainsi qu'un lien cassé pointant vers un fichier d'image "**nps-logo_mod.bmp**" non-disponible.

Lancer Firefox avec la commande **./start_firefox &** permettant d'accéder au site Web présent sur le conteneur et déposer ci-dessous une copie de la page.

Quel est l'intérêt d'utiliser le caractère '**&**' dans la commande ci-dessus ?

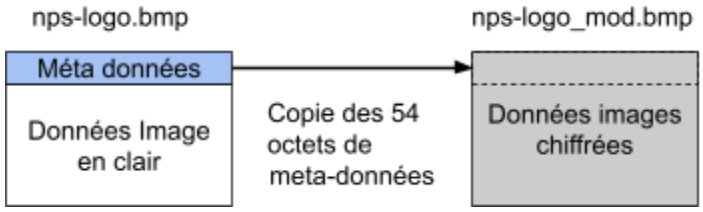
2.5.1 Chiffrement AES en mode ECB



Nous allons chiffrer le fichier nps-logo.bmp par **AES** en mode **ECB** et enregistrer le résultat dans le fichier nps-logo_mod.bmp

- **$ openssl aes-128-ecb -e -in nps-logo.bmp -out nps-logo_mod.bmp -K <clé>**
- Faire un listing du répertoire courant (**ll**) pour vérifier que le fichier chiffré est bien présent
- Lancer Firefox pour vérifier si vous pouvez visualiser le nouveau fichier image sur la page Web. Quel est le problème ?

- On va maintenant copier les métadonnées du fichier BMP d'origine vers le pour "faire croire" à notre système qu'il s'agit d'un fichier BMP classique et ainsi pouvoir visualiser le contenu via le site Web



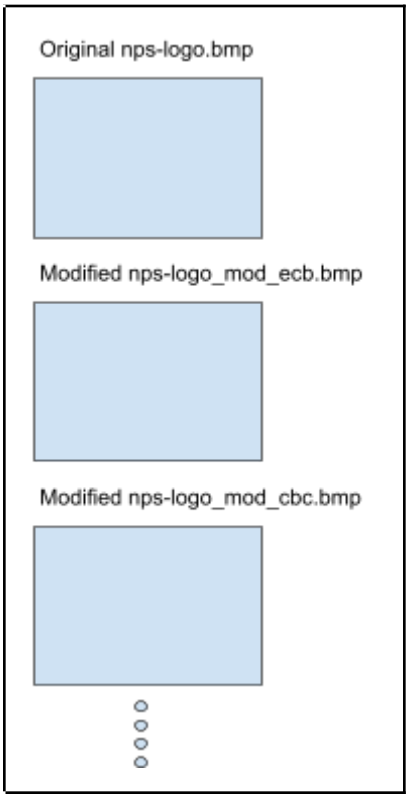
- Effectuer cette opération avec la commande :
  $ dd if=nps-logo.bmp of=nps-logo_mod.bmp bs=1 count=54 conv=notrunc

- Rafraichir le site Web (ctrl + F5) et copier le rendu du fichier image chiffré ci-dessous :

- Quel semble être le souci avec le mode de chiffrement ECB ?

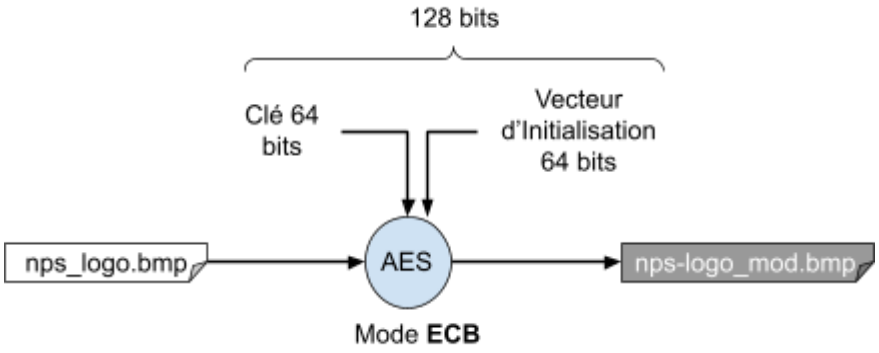
Dans cette partie, vous allez créer des chiffrement de l'image originale par des procédés plus robustes : CBC, CFB, OFB.

Pour comparer ces différents résultats, nous voulons obtenir toutes les images sur la même page Web comme illustré ci-contre →



La tâche va consister à :

1. Chiffrer le fichier original avec les différents modes (cbc, cfb ou ofb) et sauvegarder les résultats sous un autre nom (nps-logo_xxx.bmp)
2. Mettre à jour les entêtes des fichiers obtenus en les empruntant au fichier original.
3. Mettre à jour le fichier index.html pour intégrer les liens vers les nouvelles images en conservant celles déjà présentes
4. Visualiser la page et noter les différences entre les "images"



NOTE :
Contrairement à l'**ECB**, le chiffrement **AES** en modes **CBC**, **CFB, OFB** nécessite l'utilisation d'un **vecteur d'initialisation** de même longueur que la clé (64bits), ce qui portent la "clé globale" à 128 bits

Fournir un screenshot de l'intégralité de la page comportant toutes les "images" obtenues avec les différents modes de chiffrement :

Conclure sur l'apparente robustesse de tous ces modes de chiffrement :