TP2 - AES (correction)

(AES) **Exercise 1.** *BlockCipher-AES*

L'objectif de ce TP est de comprendre le standard de chiffrement par block AES (Advanced Encryption Standard). Le schéma de chiffrement symétrique AES a été publié en 1977, et a été sélectionné par le NIST (National Institute of Standards and Technology) pour devenir le standard en chiffrement symétrique par block en 2001, à la suite d'un processus de standardisation de 5 ans. Le chiffrement AES peut être assimilé (c'est le cas en pratique) à la brique de chiffrement dans les modes de chiffrement ECB, CBC, CTR, GCM et autres.

Ce chiffrement opère sur des blocs de tailles 16 octets (4 mots de 4 octets représentés sous la forme d'un tableau de 4×4 (état) où la lecture est effectuée en colonne (un mot = une colonne). AES fonctionne en répétant la même suite d'instructions un certain nombre de fois sur l'état. La première étape de l'AES consiste en l'algorithme `KeyExpansion` qui retourne autant de clés dérivées de la clé secrète k que de tours dans le chiffrement. Avant d'entrer dans la boucle, l'état est xoré avec la clé dite de *ronde*. Après ça, les *tours de ronde* se répètent en appliquant dans l'ordre bien précis 4 fonctions :

1. `SubBytes` : substitue les octets afin de casser la linéarité
2. `ShiftRows` : rotation des lignes de manière cyclique d'un nombre d'octet égale à la ligne
3. `MixColumns` : multiplie chaque colonne par une matrice cyclique
4. `AddRoundKey` : XOR de la clé de ronde avec l'état colonne par colonne

Il est important de noter que lors du dernier tour de ronde, la fonction `MixColumn` n'est pas appliquée. Il existe 3 sécurité pour l'AES, qui repose sur la taille de la clé et le nombre de tours de ronde :

	Taille de clé (N_k mots de 4 octets)	Taille des blocs (N_b mots de 4 octets)	Nombre de tours (N_r)
AES-128	4	4	10
AES-128	6	4	12
AES-128	8	4	14

Toutes les informations nécessaires au développement de l'AES se trouve dans une notice FIPS-197. Vous trouverez sans problème cette notice sur le site du NIST. De la même manière que le tp1 et ChaCha20, vous trouverez des vecteurs tests pour chacune des fonctions à la fin de la documentation (en annexe A si vous souhaitez suivre tout le processus d'un état au format tableau et comprendre certaines opérations, et en annexe C pour suivre sous forme de suite d'octet classique). La documentation interprète les colonnes des états comme des polynômes de $GF(2^8)$, soit, des polynôme dont les coefficients sont dans F_{2^8} : il suffit de comprendre que les coefficients correspondent à un octet chacun.

1. Avant de commencer, il est important de comprendre la structure des états. Implémentez une fonction `initialize_state` qui prend en entrée une suite de 16 octets et qui retourne l'état correspondant à cette suite d'octet.

2. De la même manière que la question précédente, implémentez une fonction `state2bytes` qui prend en entrée un état et qui retourne la suite d'octet correspondante.

Pour commencer, il convient de traiter d'abord chacune des fonctions que l'on a évoqué indépendamment les une des autres. L'idée est donc d'implémenter dans un premier temps la fonction `SubBytes`.

3. Implémenter la fonction `SubBytes` qui prend en entrée un état `state` et retourne la transformation de cet état par la S-box. Deux choix s'offrent à vous : vous pouvez trouver facilement la S-box de l'AES sur n'importe quel git (laissez le chat tranquille il consomme déjà bien assez d'eau et il ne vous apprendra rien), ou bien écrire la multiplication décrite dans la figure (5.2) de la documentation.

Tout l'intérêt de cette fonction réside dans l'addition de l'octet $0x63$ qui vient casser la linéarité de la transformation afin qu'elle ne soit pas simplement inversible par un produit matriciel.
Après ça, si l'on se réfère à la documentation, l'état passe par la fonction `ShiftRows`, qui s'occupe d'effectuer une rotation bien précise de chacune des lignes.

4. Implémenter la fonction `ShiftRows` qui prend en entrée un état et retourne ce même état après transformation.

La fonction `MixColumn` utilise l'interprétation polynomial afin de multiplier chaque colonne par un polynôme $a = \texttt{0x03}X^3 + \texttt{0x01}X^2 + \texttt{0x01}X + \texttt{0x02}$ et réduire modulo $X^4 + 1$. Cette notation peut faire peur mais ce n'est rien que vous ne savez pas faire : il faut comprendre une colonne $[s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}]$ comme un polynôme $f(X) = s_{0,c} + s_{1,c}X + s_{2,c}X^2 + s_{3,c}X^3$. La multiplication par un polynôme suivie de la réduction modulo $X^4 + 1$ est interprétée comme une multiplication à gauche par une matrice cyclique :

$$\begin{pmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{pmatrix}$$

Il est important de noter que dans ce cas, l'addition présente lors de la multiplication matrice-vecteur n'est pas l'addition classique de $\mathbb{Z}$, mais bien le XOR puisque l'on est dans $GF(2^8)$.

5. Implémenter la fonction `MixColumns` qui prend en entrée un état et retourne ce même état après transformation.

Finalement, la fonction `AddRoundKey` conclu le tour de ronde en xorant la clé de ronde avec l'état colonne par colonne.

6. Implémenter la fonction `AddRoundKey` qui prend en entrée un état et la clé de ronde et retourne ce même état après son XOR avec la clé.

Le principe étant de répéter ces tours de ronde, alors il suffit maintenant d'emboiter dans le bon ordre ces 4 fonctions.

7. Implémenter la fonction `Round` qui prend en entrée un état et une clé de ronde et retourne ce même état après un tour complet de transformation.

À partir d'ici, votre AES est prêt ! En omettant la fonction `KeyExpansion` que l'on ne traitera pas ici, on peut désormais construire la fonction de chiffrement d'un bloc par AES.

8. Implémenter la fonction `Cipher` qui prend en entrée un état `state` de taille 16 octets, une suite de $N_r + 1$ clés de ronde de taille 16 octets et retourne le chiffré correspondant.

Bonus : Si vous souhaitez aller plus loin, vous pouvez implémenter l'inverse de chacune des fonctions précédente : `SubBytes, ShiftRows, MixColumn, AddRoundKey`, afin d'implémenter la fonction `Decrypt` qui prend en entrée un chiffré `ciphertext` et une suite de clé de ronde `w`, et qui retourne le `plaintext` sous jacent.