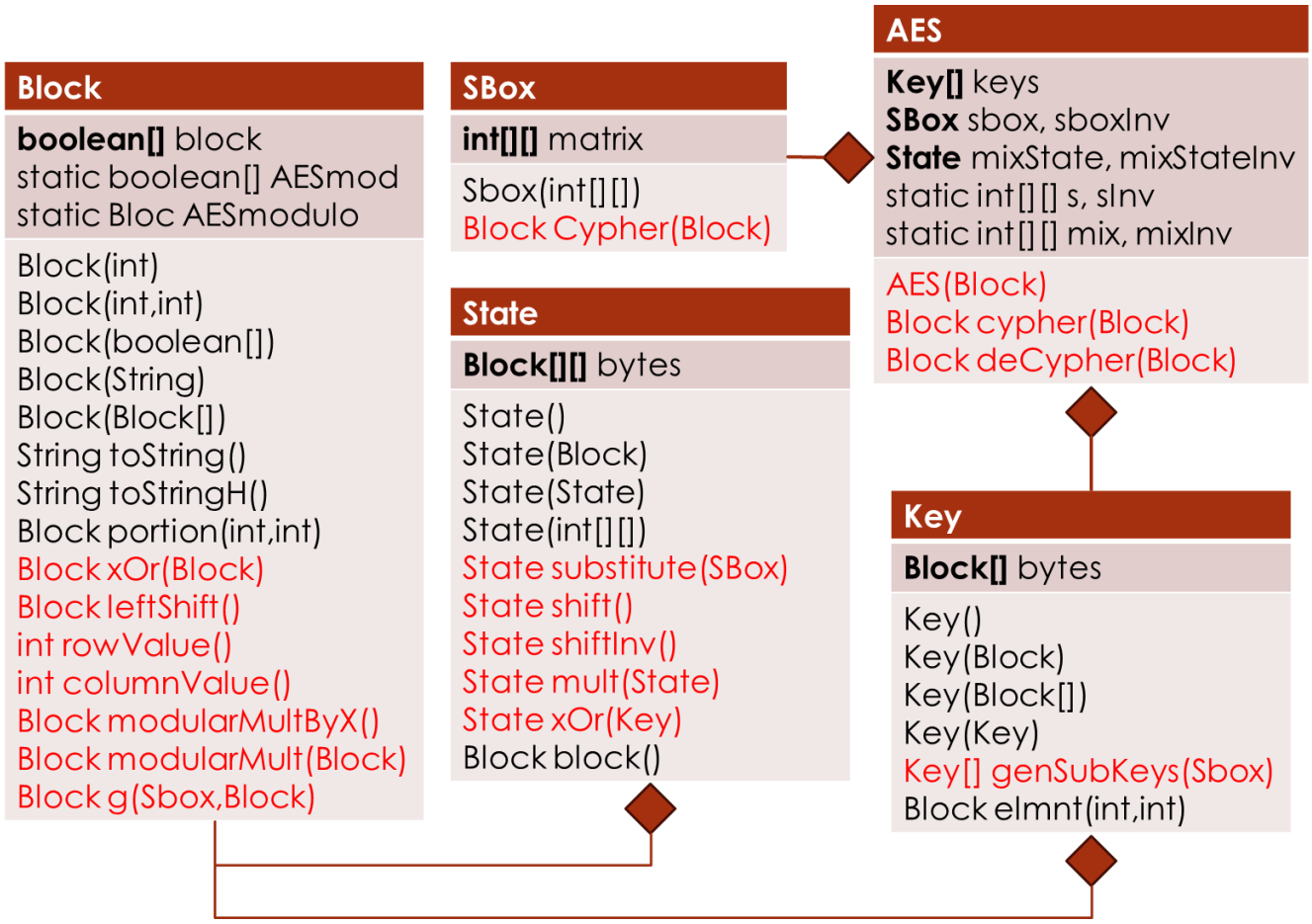
Sécurité des réseaux TP2

Le TP consiste à implémenter l'algorithme de chiffrement AES. Vous trouverez une description exhaustive de cet algorithme dans le fichier Chap5NetworkSecurity.pdf qui se trouve sur mon site.

Voici le diagramme UML des classes du projet :



I) Classe Block

La classe *Block* permet de représenter des séquences de bits. Les bits de cette séquence seront stockés dans l'attribut block qui est un tableau de booléen (un booléen par bit). Les objets de cette classe permettront de représenter les blocks de données à chiffrer, les octets de données utilisés par les états de AES, ainsi que les clés et les sous-clés. Cette classe comporte un membre statique, appelé AESmodulo, qui est lui-même un bloc. Il s'agit d'un bloc de 8 bits (un octet) qui représente le polynôme irréductible utilisé comme modulo dans l'arithmétique de $GF(2^8)$. Pour être plus précis, il s'agit des valeurs des coefficients de ses monômes de degré 0 à 7 ($x^4 + x^3 + x + 1$), et le coefficient du monôme de degré 8, qui vaut 1, n'est pas représenté car il n'est pas nécessaire dans les calculs.

Cette classe comporte plusieurs constructeurs. Le premier construit un *Block* à partir de sa taille, sans initialiser les valeurs des bits (ils sont par défaut mis à 0), et le second à partir de sa taille, et de la valeur (un entier) qu'il doit coder (en binaire). Le troisième construit un *Block* à partir d'un tableau de booléens qui contient les valeurs des bits. Le quatrième construit un bloc à partir d'une chaine de caractères représentant la séquence de bits par des caractères '0' et '1'. Enfin, le dernier constructeur prend en paramètre un tableau de *Block* et va créer un nouveau block qui concatène les différents blocks du tableau.

La fonction *toString* retourne une chaine de caractère décrivant la valeur des différents bits du bloc. La fonction *toStringH* va également retourner une chaine de caractère décrivant les différents bits, mais à la différence de la fonction *toString*, la description est donnée en hexadécimal.

La fonction *portion* va renvoyer un sous-bloc du bloc. On considère que le bloc est découpé en un certain nombre de sous-blocs de même taille, où le nombre de sous-blocs correspond au premier entier pris en paramètre. Le second paramètre entier indique lequel de ces sous-blocs est renvoyé par la fonction. Ainsi, si le bloc correspond à la séquence de bits 011001, alors portion(2, 0) appliqué à cette séquence correspondra à la première moitié 011, portion(2, 1) à la seconde moitié 001, portion(3, 0) au premier tiers 01, portion(3, 1) au second tiers 10, et ainsi de suite.

La fonction *xOr* (à implémenter) va retourner un nouveau bloc qui correspond à l'application du XOR bit à bit entre le block appelant et le bloc donné en paramètre.

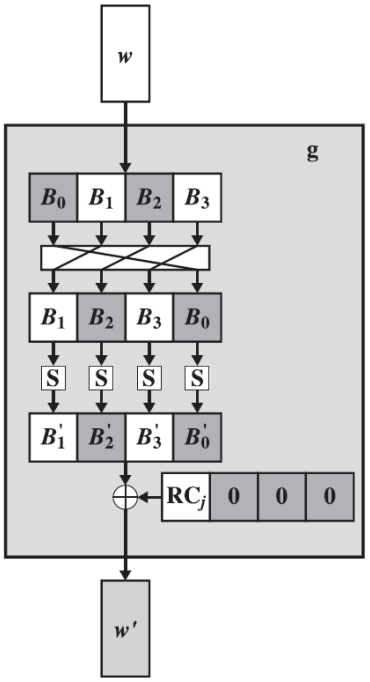
La fonction *leftShift* (à implémenter) va retourner un nouveau block qui correspond à un décalage à gauche du block appelant d'un bit. Cette fois, le décalage n'est pas circulaire, et le bit qui se trouvait le plus à gauche est perdu. Le bit le plus à droite du nouveau bloc aura la valeur 0.

Les fonctions *rowValue* et *columnValue* (à implémenter) renvoient les numéros de ligne et de colonne dans la matrice de la S-box correspondant au block appelant. Ces fonctions pourront être utilisées lors de l'implémentation d'une S-box. Attention, ces fonctions n'ont de sens que si la taille du block est de 8 bits, comme c'est le cas des blocks en entrée d'une S-box.

La fonction *modularMultByX* (à implémenter) renvoie un *Block* dont la valeur correspond aux coefficients du polynôme résultat de multiplication par x du polynôme dont les coefficients sont ceux du *Block* appelant. Notez que cette multiplication correspond à celle définies dans $GF(2^8)$, et qui s'effectue modulo le polynôme irréductible $x^8 + x^4 + x^3 + x + 1$. Vous pouvez utiliser la formule vue en cours (chapitre 6, slide 7) pour effectuer cette multiplication. Dans ce cadre, vous pourrez utiliser les fonctions *leftShift* et *xOr*, ainsi que l'attribut statique *AESmodulo*.

La fonction *modularMult* (à implémenter) va effectuer la multiplication entre le bloc appelant et le *Block* pris en paramètre. Cette multiplication est également effectuée en utilisant l'arithmétique de $GF(2^8)$, en considérant les blocs comme des polynômes, et en effectuant la multiplication modulo le polynôme irréductible. Vous pourrez utiliser les fonctions *modularMultByX* et *xOr* pour effectuer l'opération plus facilement.

La fonction *g* (à implémenter) est la fonction utilisée lors de la génération de clé, et qui porte le même nom dans le document Chap5NetworkSecurity.pdf (voir p. 149, ainsi que dans le schéma à droite). Elle prend en paramètre la SBox qui va être appliquée aux octets, et qui sera la même que celle appliquée dans l'opération de chiffrement « substitutes bytes ». Le second paramètre est l'octet représentant



$RC[j]$, où j correspond au nombre de fois où cette fonction a été appelée lors de la génération. Notez que le polynôme représentant cette valeur est $x^j mod\ m(x)$, où $m(x)$ est le polynôme irréductible décrit plus haut. La valeur de $RC[1]$ est par exemple 00000010, celle de $RC[2]$ est 00000100. Notez également que cette valeur sera définie par la fonction appelantes (qui sera la fonction *genSubKey* de la classe *Key*), et que lors de l'implémentation de cette fonction vous n'avez pas à vous en soucier.

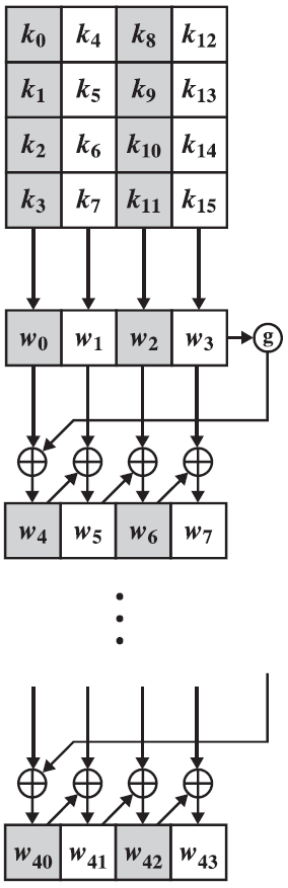
II) Classe SBox

Il s'agit de la classe qui permet de modéliser la S-box présentées dans Chap5NetworkSecurity.pdf. Cette classe contient comme attribut la matrice d'entiers qui permet le chiffrement d'un bloc de 8 bits en un autre bloc de 8 bits. Les indices de ligne et de colonne correspondant à un bloc de 8 bits peuvent être obtenus à l'aide des fonctions *rowValue* et *columnValue*.

Le constructeur permet d'initialiser les éléments de la matrice à partir d'une matrice d'entiers donnée en paramètre. Les matrices des deux S-boxes utilisées, pour le chiffrement et le déchiffrement, sont contenues dans les attributs statiques *s* et *sInv* de la classe *AES* qu'on verra plus tard.

La fonction *cypher* (à implémenter) va renvoyer le block de 8 bits obtenu par la S-box pour le bloc de 8 bits pris en paramètre. Pour cela, il faut aller chercher dans la matrice une valeur, qui sera ensuite traduite en binaire pour obtenir une séquence de bits, et qui sera renvoyée sous la forme d'un bloc.

III) Classe Key



Il s'agit de la classe qui permet de modéliser une clé ou une sous-clé dans l'algorithme AES. Les clés et les sous-clés sont composées de 4 blocs de 4 octets, qui sont notés w_i dans Chap5NetworkSecurity.pdf. Chacun de ces blocs de 4 octets est représenté par un objet de type *Block*. Ces quatre blocs sont stockés dans l'attribut *bytes*, qui est un tableau de *Bloc*.

La classe comporte un certain nombre de constructeurs. Le premier ne prend pas de paramètre et va créer une clé dont tous les bits sont à 0. Le second va prendre en paramètre un *Bloc*, qui va être une représentation en binaire des 128 bits de la clé, et créer l'objet *Key* correspondant. Le troisième constructeur prend en paramètre un tableau de blocs, contenant les 4 blocs de 32 bits de la clé, et créer l'objet *Key* correspondant. Enfin le dernier constructeur prend en paramètre une clé (objet de la classe *Key*), et va créer une copie de cette clé.

La fonction *genSubKey* (à implémenter) va générer l'ensemble des sous-clés utilisées lors de AES, à partir de la clé appelante, et les renvoyer sous la forme d'un tableau de *Key*. Ce tableau doit comprendre 11 clés distinctes, où la première clé est une copie de la clé appelante, et où les clés suivantes sont générées à partir de la clé appelante. La description de l'algorithme pour générer les clés se trouve dans le fichier Chap5NetworkSecurity.pdf (p. 149, ainsi que dans le schéma à droite). Vous pourrez utiliser la fonction *g* et la fonction *xOr* de la classe *Block* pour implémenter cette fonction.

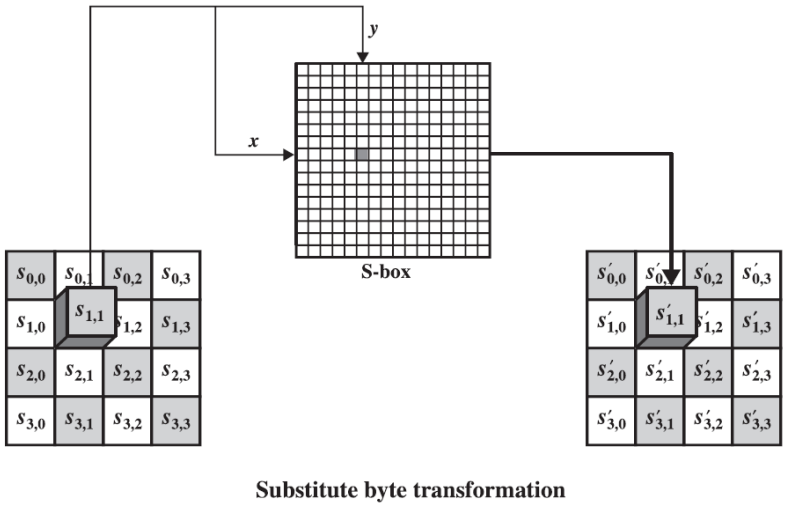
Enfin la fonction *elmnt* va être utilisée pour accéder aux éléments de la clé, comme si elle était représentée comme un état, c'est-à-dire une matrice 4×4 d'octets. Cette fonction prend en paramètre deux entiers, qui indiqueront le numéro de ligne et de colonne, et le *Bloc* renvoyé sera l'octet qui se trouve à cette position dans la représentation matricielle de la clé.

IV) Classe State

Il s'agit de la classe qui permet de modéliser un état. Un état est une représentation matricielle (4×4) du bloc à chiffrer. Les éléments dans la matrice sont des *Blocs*, qui représentent des octets (séquence de 8 bits). Ces *Blocs* sont contenus dans l'attribut *bytes*, qui est un tableau à deux dimensions.

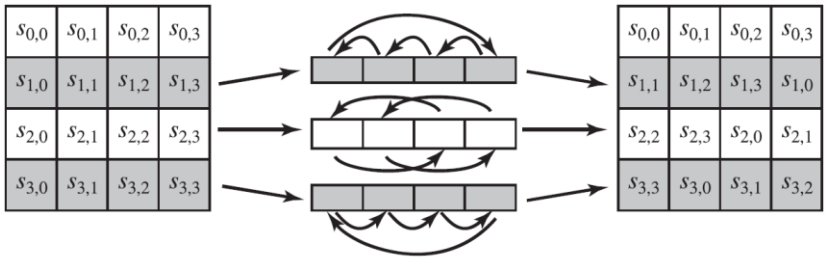
Cette classe comporte un certain nombre de constructeurs. Le premier n'a pas d'argument et va initialiser la valeur de l'ensemble des blocs à 0. Le second prend en paramètre un *Bloc*, qui devra contenir 128 bits, et le constructeur va remplir les éléments de la matrice à partir des bits du bloc, en utilisant la répartition décrite dans l'algorithme AES. Le troisième constructeur va créer une copie d'un *Bloc*. Enfin, le dernier constructeur va créer un état à partir d'une matrice d'entiers qui décrit en décimal les différents éléments de l'état. Ce constructeur sera utilisé pour créer la matrice (qui sera un *State*) qui sera multipliée à l'état courant lors de l'opération « mix column ». Il devra donc être appelé pour construire la matrice utilisée lors du chiffrement, avec en paramètre l'attribut statique *mix* de la classe AES. Il devra également être appelé pour construire la matrice utilisée lors du déchiffrement, avec en paramètre l'attribut statique *mixInv* de la classe *AES*.

La fonction *substitute* (à implémenter) va effectuer l'opération « substitute bytes » (décrite p.138 dans Chap5NetworkSecurity.pdf) au bloc courant, et renvoyer le résultat sous la forme d'un nouvel état. Cette fonction va utiliser la *SBox* prise en paramètre pour cette opération de substitution. La fonction pourra indifféremment prendre la S-box en paramètre lors du chiffrement, et S-box inverse lors du déchiffrement.



Substitute byte transformation

Les fonctions *shift* et *shiftInv* (à implémenter) vont effectuer respectivement les opérations « shift rows » et « inverse shift rows », et renvoyer le résultat sous la forme d'un *Bloc*. Ces opérations sont décrites dans Chap5NetworkSecurity.pdf (p. 143), et consistent à décaler les octets présents dans les lignes de la matrice de l'état.


Shift row transformation

La fonction *mult* (à implémenter) va effectuer la multiplication matricielle (en utilisant l'arithmétique de $GF(2^8)$) entre l'état appelant, et l'état pris en paramètre, et renvoyer le résultat sous la forme

d'un *Block*. Cette multiplication est utilisée lors de l'opération « mix columns ». Vous pourrez utiliser les fonctions *xOr* et *modularMult* de la classe *Block* pour implémenter cette fonction.

La fonction *xOr* (à implémenter) va effectuer l'opération du OU exclusif entre l'état appelant et la clé prise en paramètre, et renvoyer le résultat sous la forme d'un *Bloc*. C'est cette fonction qui sera utilisé lors de l'opération « add round key ». L'opération va s'effectuer sur la représentation de la clé sous la forme d'un état. Vous pourrez accéder au bloc de cette représentation de la clé en utilisant la fonction *elmnt* de la classe *Key*, qui prend en paramètre deux entiers correspondant aux numéros de ligne et de colonne de l'octet à extraire de la clé (valeurs entre 0 et 3 pour les deux entiers). Vous pourrez également utiliser la fonction *xOr* de la classe *Bloc* pour effectuer l'opération sur chaque octet.

Enfin, la fonction *block* va renvoyer le bloc de données (codé sur 128 bits) correspondant à l'état.

V) Classe AES

La classe *AES* est celle qui va appliquer l'algorithme de chiffrement/déchiffrement AES à des blocs de données. Elle contient cinq attributs, qui sont des éléments similaires à chaque appel de l'algorithme de chiffrement et de déchiffrement pour une clé donnée. Le premier attribut *keys* est un tableau de *Key*, contenant les différentes sous-clés utilisées lors de l'algorithme. Les deux attributs suivants *sbox* et *sboxInv* sont deux S-boxes (objets de type *SBoxes*) qui seront utilisées lors du chiffrement et déchiffrement. La S-box *sboxInv* sera l'inverse de la S-box *sbox*, et ces deux S-boxes sont décrites dans Chap5NetworkSecurity.pdf (p. 139). Elles sont utilisées par les opérations « substitution bytes », « inverse substitution bytes » et dans la fonction *g* lors de la génération de sous-clés. Les deux derniers attributs *mixState* et *mixStateInv* sont deux matrices d'octets 4×4, représentées par des objets de la classe *State*. Ces deux matrices vont être utilisées lors des opérations « mix columns » et « inverse mix column ». Ces deux matrices sont décrites dans Chap5NetworkSecurity.pdf (p. 144).

La classe *AES* contient également un certain nombre d'attributs statiques qui correspondent aux valeurs des S-boxes *sbox* et *sboxInv* (*s* et *sInv*), ainsi que les coefficients des matrices (qui sont représentées par des *State*) *mixState* et *mixStateInv* (*mix* et *mixInv*). Ces attributs statiques pourront être directement utilisés par les constructeurs de la classe *SBox* (pour les deux S-boxes) ou de la classe *State* (pour les deux matrices).

Le constructeur (à implémenter) va prendre en paramètre la clé de chiffrement (représentées par un *Block*) codée sur 128 bits. A partir de cette clé de chiffrement, le constructeur va calculer les 11 sous-clés utilisées lors du chiffrement/déchiffrement. Le constructeur va également créer les deux S-boxes en initialisant les attribut *sbox* et *sboxInv*, ainsi que les deux matrices *mixState* et *mixStateInv*.

La fonction *cypher* (à implémenter) va effectuer le chiffrement d'un bloc de données pris en paramètre, et renvoyer le bloc de données résultant de ce chiffrement. Le chiffrement est effectué avec la clé qui avait été donnée en paramètre au constructeur, donc on peut directement utiliser toutes les sous-clés stockées dans l'attribut *keys*. La fonction *deCypher* (à implémenter) va effectuer le même type de travail, mais pour le déchiffrement. Notez que cette fois, et à l'inverse de DES, le code de ces deux fonctions ne sera pas similaire.

VI) Travail à effectuer

Vous devez implémenter l'ensemble des fonctions indiquées en rouge dans le diagramme UML. Vous pourrez utiliser la fonction *main* de la classe *AES* pour tester votre code. Le bloc de données, ainsi que la clé, correspondent à l'exemple donnée dans le document Chap5NetworkSecurity.pdf (voir p. 151, les résultats des différentes étapes de calcul sont donnés dans ce document pour cet exemple). Le déchiffrement doit permettre de retourner au bloc de données original.