# TP Architecture des réseaux

*Ce TP s'effectue en langage Java.*

## I) Simplified DES

L'objectif est de programmer une version simplifiée de l'algorithme DES. Cette version simplifiée de l'algorithme se trouve sur ma page web. Vous trouverais également sur ma page web un canevas de la classe DES à remplir, ainsi que les classes complémentaires Block et Key. Vous devez donc remplir l'intégralité des fonctions du fichier DES.java afin que l'algorithme puisse coder et décoder.

La classe Block permet de gérer des blocks de données contenant 8 bits. Elle sera à la fois utilisés pour les blocks de données de 8 bits générés par l'algorithme (pas uniquement le plaintext et le cyphertext), ainsi que les sous-clés (générées à partir de la clé). Le constructeur sans paramètre construit une block de 8 zéros consécutifs, et le constructeur prenant une chaine de caractères en paramètre construit un block correspondant à la chaine de caractère (une exception est lancée si le format n'est pas correct).

La classe Key est utilisé pour représenter les clés qui sont codées sur 10 bits. Le constructeur sans paramètre construit une clé composée de 10 zéros consécutifs, et le constructeur prenant une chaine de caractères en paramètre construit une clé correspondant à la chaine de caractère (une exception est lancée si le format n'est pas correct).

## II) Liste des méthodes à compléter

**DES(Block block,Key key):** constructeur qui se contente de créer un instance à partir des deux paramètres.

**P10(Key key):** applique la permutation P10 sur la clé (sans la modifier) et renvoie le résultat

**P8(Key key):** applique la permutation P8 sur la clé (sans la modifier) et renvoie le résultat sous la forme d'un object de type Block (8 bits)

**LS1(Key key):** effectue séparément le décalage à gauche d'un bit sur les deux blocks de 5 bits de la clé (sans la modifier) et renvoie le résultat

**LS2(Key key):** effectue séparément le décalage à gauche de deux bits sur les deux blocks de 5 bits de la clé (sans la modifier) et renvoie le résultat

**ArrayList KeyGen():** génère les deux sous-clés qui sont stockées dans une ArrayList qui est retournés par la fonction

**Block IP(Block block):** applique la permutation IP sur le block (sans le modifier) et renvoie le résultat

**Block FirstHalfChange(Block block,boolean[] halfBlock):** remplace les 4 premiers bits de block par les 4 bits de halfBlock

**Block IPinv(Block block):** applique la permutation inverse de IP sur le block (sans le modifier) et renvoie le résultat

**Block EP(Block block):** applique l' extension/permutation E/P du block (sans le modifier) et renvoie le résultat

**Block XOR8(Block block,Block key):** effectue un XOR entre les deux blocks et renvoie le résultat

**boolean[] XOR4(Block block,boolean[] halfBlock):** effectue un XOR entre deux tableau de booléens de taille 4 et renvoie le résultat

**int BoolToInt(boolean b0,boolean b1):** donne la valeur entière correspondant à $2 \times b0 + b1$

**boolean IntToBool0(int i):** donne la valeur booléenne du bit le plus à gauche de la représentation binaire de i (par exemple si i=2, la représentation binaire est 01 et la valeur retournée est 0)

**boolean IntToBool1(int i):** donne la valeur booléenne du bit le plus à droite de la représentation binaire de i (par exemple si i=2, la représentation binaire est 01 et la valeur retournée est 1)

**boolean[] S(Block block):** applique la fonction S0 aux 4 premiers bits de block, et la fonction S1 aux 4 derniers bits de block, et renvoie le résultat sous la forme d'un tableau de 4 booléens (les 2 premiers booléens correspondent à S0, et les 2 suivants à S1)

**boolean[] P4(boolean[] halfBlock):** applique la permutation P4 sur un tableau de booléens de taille 4 (sans le modifier) et renvoie le résultat

**Block SW(Block block,boolean[] halfBlock):** échange les 4 premier bits de block avec les 4 derniers, et remplace les 4 derniers bits de block par les valeurs de halfBlock (sans changer le block) et renvoie le résultat décrite

**boolean[] FK(Block block,Block key):** applique la function $f_K$ décrite dans l'annexe

**Block Encode:** effectue le chiffrement du block de l'instance en utilisant la clé de l'instance et renvoie le résultat sous la forme d'un block

**Block Decode(Block block):** effectue le déchiffrement du block (en paramètre) en utilisant la clé de l'instance et renvoie le résultat sous la forme d'un block