Compte rendu Codage | DUBOIS - FARIZON

TP1-A

Hadamard :

Nous avons fait le code de l'ensemble des fonctions du codage de Hadamard tout en ajoutant la technique des séquences d'étalement.
Pour ce faire, nous avons créer un fichier annexe dans lequel il y a les fonctions les plus « basiques » sur les matrices. Il y a une fonction d'affichage, d'initialisation et d'inversement des valeurs des cases d'une matrice passée en paramètre.
Ensuite pour ce qui concerne le codage de Hadamard, on descend jusqu'à la matrice de taille minimale puis on remonte en renvoyant une matrice de la taille supérieure à chaque fois. Et pour l'agrandissement et le remplissage de la matrice, on fait des recopies dans les différents coin de la matrice en reprenant la précédente pour copie.

En ce qui concerne les séquences d'étalements nous avons repris ce que nous avions vu en cours et en TD en implémentant les différentes étapes. Nous avons utilisé une structure dans laquelle il y a 3 tableaux pour stocker les 3 bits choisis aléatoirement pour le calcul de la séquence, pour la ligne qui correspond à l'utilisateur et enfin pour la séquence finale après calculs. Et pour le reste du code c'est une traduction de toutes les manipulations vues en TD.

Nous avons decider d'utiliser un tableau afin de stocker les messages entre chaque fonctions.
La fonction coderH sert a transformer un message claire en un message codé a partir d'une ligne de la matrice d'Hadamard.
Pour cette fonction nous utilisons six paramètres :
- « messageACoder » qui est le message que l'on va codé, sous forme de tableau,
- « messageCoder » qui est le message une fois codé, sous forme de tableau,
- « matriceH » qui est la matrice d'hadamard qui sera utilisé pour le codage du message,
- « ligne » qui est le numéro de la ligne de la matrice d'hadamard,
- « tailleMessage » qui donne le nombre de bits contenu dans le message une fois codé,
- « longueur » qui est la taille d'une ligne de la matrice d'hadamard utilisé.

Dans cette première fonction nous avons deux boucles. Une premiere boucle grace a laquelle on passe au bit suivant a codé. Ainsi qu'une seconde qui va nous permettre de lire la ligne de la matrice et d'écrire le résultat dans le tableau du message codé.

La fonction decoderH, elle, sert a faire l'operation inverse, décoder un message a partir d'une ligne de la matrice d'Hadamard.
Pour cette fonction nous utilisons six paramètres :
- « messageCode » qui est le message codé, sous forme de tableau,
- « messageClair » qui est le message une fois décodé, sous forme de tableau,
- « matriceH » qui est la matrice d'hadamard qui sera utilisé pour le codage du message,
- « ligne » qui est le numéro de la ligne de la matrice d'hadamard,
- « tailleMessage » qui donne le nombre de bits contenu dans le message une fois codé,
- « longueur » qui est la taille d'une ligne de la matrice d'hadamard utilisé.

Dans la deuxième fonction nous avons une première boucle qui initialise toutes les cases du tableau qui contiendra le message de base à la valeur 0.
Le deuxième boucle fait juste une copie du message codé dans au autre tableau sur lequel on fera le traitement. Nous avons choisi de faire cela pour toujours garder une trace du message codé qui a été donné en paramètre en cas de problème au sein de la fonction.
La troisième grande boucle effectue le traitement de cette copie. Elle traite un chiffre pour le message clair par tour. Le bit de code est multiplié par la ligne de la matrice qui a servi pour l'encodage puis les bits de code qui correspondent a un bit du message clair sont additionnés. Ce résultat sera ensuite divisé par la longueur de la ligne de la matrice d'hadâmard utilisée pour donné le bit du message clair qui sera ensuite stocker dans un tableau contenant tout les bits du message clair.

TP1-B

Arithmétique :

Nous avons choisi de faire le codage arithmétique en suivant l'ordre alphabétique et non l'ordre d'apparition des lettres dans le message.
Nous avons choisi d'utiliser un tableau pour stocker le message et un deuxième pour stocker les lettres que l'on trouvera au fur et a mesure de la lecture du message avec leur nombre d'occurence.
Une premiere boucle va nous permettre d'initialiser le tableau qui contiendra le nombre d'occurence pour chaque lettre. On utilise le code ASCII de la lettre pour la retrouver.
Dans ce même tableau, avec la deuxième boucle nous allons transformer ce nombre d'occurrences et fréquence d'apparition.
Ensuite, avec un autre tableau, a seulement deux cases, nous allons stocker les bornes.
Une premiere partie va nous permettre d'abord recopier dans un nouveau tableau seulement les lettres utilisées pour le calcul des bornes supérieur des intervalles, il est donc plus petit que le premier.
La ligne suivante permet de calculer les bornes supérieurs pour chaque lettres du message.
Ensuite on recopie ces valeurs dans le premier tableau.
La dernière partie du code sert a faire le calcul de l'intervalle final.

HDBN : Nous avons appliqué ce que nous avions vu en cours avec les derniers viols et les changements de valeur en fonction de ces derniers et des bits précédents.
Nous avons suivis ces règles :
-Si c'est un 1 alors on le met à l'inverse du dernier 1
-Si c'est un 0 alors on regarde s'il y a n 0 qui se suivent
-Si non, on ne touche pas aux 0
-Si oui, on applique le codage bipolaire simple

Et avant de faire ça on a transformé le message passé en paramètre en un tableau d'entier (binaire) pour que ce soit plus facile à utiliser pour la suite.