Fiche de TP N° 3 du Module Cryptgraphie

Implémentation d'un algorithme de chiffrement par bloc (XTEA)

EXtended Tiny Encryption Algorithm (ou XTEA) est un algorithme de chiffrement par bloc connu pour la simplicité de sa description et de son implémentation (généralement quelques lignes de codes). Il s'agit d'un réseau de Feistel comprenant un nombre important de tours : 32. Il fut conçu par David Wheeler et Roger Needham, du laboratoire informatique de Cambridge, et présenté au salon *Fast Software Encryption* en 1994[1]. Il n'est l'objet d'aucun brevet.

Pour plus d'infromation , https://fr.wikipedia.org/wiki/XTEA.

Objectifs du TP:

1- Implémentation de l'algorithme XTEA (clé sur128bit et la taille des blocs en claires sur 64bit), en entré le programme lie une clé de 128bit(16 octet), et un bloc de 64bit (8octet) et donne en sortie le bloc de64bit chiffré. La fonction inverse de déchiffrement doit aussi être implémenté. L'implémentation doit être faite en JAVA.
2- Extention du chiffrement avec plusieurs modes opératoires pour chiffré un fichier quelconque (ECB,CBC, OFB, CTR et XTS).

```
public class XTEA {

    /**

     * Prend 64 bits de données de v[0] et v[1], et 128 bits de key[0] à key[3]

     */

    public static void encipher(int numRounds_U, int[] v_U, int[] key_U) {

        int v0_U = v_U[0], v1_U = v_U[1], sum_U = 0, delta_U = 0x9E3779B9;

        for (int i_U = 0; i_U < numRounds_U; i_U++) {

            v0_U += (v1_U << 4 ^ v1_U >>> 5) + v1_U ^ sum_U + key_U[sum_U & 3];

            sum_U += delta_U;

            v1_U += (v0_U << 4 ^ v0_U >>> 5) + v0_U ^ sum_U + key_U[sum_U >>> 11 & 3];

        }

        v_U[0] = v0_U;

        v_U[1] = v1_U;
```

```
    }

public static void decipher(int numRounds_U, int[] v_U, int[] key_U) {

        int v0_U = v_U[0], v1_U = v_U[1], delta_U = 0x9E3779B9, sum_U = delta_U *
numRounds_U;

        for (int i_U = 0; i_U < numRounds_U; i_U++) {

            v1_U -= (v0_U << 4 ^ v0_U >>> 5) + v0_U ^ sum_U + key_U[sum_U >>> 11 & 3];

            sum_U -= delta_U;

            v0_U -= (v1_U << 4 ^ v1_U >>> 5) + v1_U ^ sum_U + key_U[sum_U & 3];

        }

        v_U[0] = v0_U;

        v_U[1] = v1_U;

    }

}
```