TP N°02

CHIFFREMENT AUDIO TOTAL EN UTILISANT DES

Dans ce TP nous allons voir comment chiffrer un bloc de 64 bits en utilisant l'algorithme de chiffrage DES.

Le TP sera réalisé en C++ sous l'environnement de développement Visual Studio.

Pour cela, nous allons suivre les étapes suivantes :

1. Préparer des fonctions utilitaires pour nous aider dans notre travail. On aura besoin de fonction d'affichage, de conversion, et du calcul du XOR.
    - Comme le C++ n'offre pas des opérations qui manipulent le binaire, nous allons utiliser des tableaux booléens qui vont contenir les octets binaires. Pour cela, nous avons besoin des fonctions :

    `binToBool8(unsigned char binVal, bool boolVal[8])` pour convertir un binaire de 32 bits en un bloc de 32 booléens.

    `binToBool64(unsigned char binVal[8], bool boolVal[64])` pour convertir un binaire de 64 bits en un bloc de 64 booléens.

    `boolToBin64(bool boolVal[64], unsigned char binVal[8])` Pour convertir un bloc de 64 booléens en un binaire de 64 bits.
    - Définir deux fonctions pour établir le XOR entre deux blocs :

    `XOR32(bool outBloc[32], bool inBloc[32])`: Pour établir l'addition bit par bit modulo 2 entre deux blocs de 32 bits. (`outBloc = outBloc XOR inBloc`).

    `XOR48(bool outBloc[32], bool inBloc[32])`: Pour établir l'addition bit par bit modulo 2 entre deux blocs de 48 bits. (`outBloc = outBloc XOR inBloc`).
    - Définir des fonctions d'affichage de blocs de booléens :

    `afficheBoolBloc64(bool boolBloc[64])` : Pour afficher le contenu d'un bloc de 64 booléens sous forme binaire.

    `afficheBoolBloc56(bool boolBloc[56])` : Pour afficher le contenu d'un bloc de 56 booléens sous forme binaire.

    `afficheBoolBloc48(bool boolBloc[48])` : Pour afficher le contenu d'un bloc de 48 booléens sous forme binaire.
    - Définir une fonction d'affichage de blocs d'octets :

    `afficheBinBloc8(unsigned char binBloc[8])` : Pour afficher le contenu d'un bloc de 8 octets en hexadécimal.
2. Définir une classe appelée DES contenant des méthodes pour effectuer les différentes étapes de l'algorithme de chiffrement DES.
    - `fonctionIP(bool G[32], bool D[32], bool inBloc[64])`: Pour appliquer la permutation initiale IP en utilisant la table :

```
static int table_DES_IP[64] = {
     58,50,42,34,26,18,10,2,
     60,52,44,36,28,20,12,4,
     62,54,46,38,30,22,14,6,
     64,56,48,40,32,24,16,8,
     57,49,41,33,25,17,9,1,
     59,51,43,35,27,19,11,3,
     61,53,45,37,29,21,13,5,
     63,55,47,39,31,23,15,7
};
```

- `fonctionE(bool outE[48], bool D[32])`: Prend en entrée un bloc de 32 bits et donne en sortie un bloc de 48 bits en utilisant la table de sélection `table_DES_E` :

```
static int table_DES_E[48] = {
     32, 1, 2, 3, 4, 5,
     4, 5, 6, 7, 8, 9,
     8, 9, 10, 11, 12, 13,
     12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21,
     20, 21, 22, 23, 24, 25,
     24, 25, 26, 27, 28, 29,
     28, 29, 30, 31, 32, 1
};
```

- `fonctions_Selection(int k, bool output[4], bool input[6])`: Représente la fonction de sélection S_k, elle prend en entrée un bloc de 6 bits et en sortie un bloc de 4 bits en utilisant la table :

```
static int table_DES_S[8][4][16] = {
/* table S0 */
{{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
{0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
{4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
{15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}},
/* table S1 */
{{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
{3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
{0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
{13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}},
/* table S2 */
{{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
{13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
{13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
{1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}},
/* table S3 */
{{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
{13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
```

```
{10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
{3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}},
/* table S4 */
{{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
{14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
{4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
{11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}},
/* table S5 */
{{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
{10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
{9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
{4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}},
/* table S6 */
{{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
{13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
{1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
{6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}},
/* table S7 */
{{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
{1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
{7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
{2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}}
};
```

- `fonctionP(bool output[32], bool input[32])`: Fonction de permutation P qui utilise la table:

```
static int table_DES_P[32] = {
     16, 7, 20, 21,
     29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2, 8, 24, 14,
     32, 27, 3, 9,
     19, 13, 30, 6,
     22, 11, 4, 25
};
```

- `fonctionF(bool outF[32], bool R[32], bool roundKey[48])`: Pour effectuer la fonction de chiffrement f qui prend en entrée un bloc de 32bits et la clé sélectionnée par la fonction de planification KS (Key Schedule) sur 48 bits, et donne en sortie un bloc de 32bits. On utilise les fonctions : `fonctionE`, `XOR48`, `fonctions_Selection`, `fonctionP`.
- `PC1(bool permutedKey[56], bool key[64])`: Pour établir le choix permuté PC-1 en utilisant la table :

```
static int table_DES_PC1[56] = {
     57, 49, 41, 33, 25, 17, 9,
     1, 58, 50, 42, 34, 26, 18,
```

```
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
};
```

- `PC2(bool keyout[48], bool key[56])`: Pour établir le choix permuté PC-2. Pour convertir une clé de 56 bits en 48 bits en utilisant la table :

```
static int table_DES_PC2[48] = {
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
};
```

- `rotationGauche(bool permutedKey[56])` : Pour établir un décalage à gauche avec rotation.
- `KS(bool key[64], bool K[16][48])`: Fonction de planification KS (Key Schedule) qui prend en entrée une clé de chiffrement de 64bits, et donne en sortie 16 clés de 48 bits. Cette fonction fait appel aux fonctions `PC1`, `rotationGauche` et `PC2`.
- `fonctionIP_inv(bool outIPinv[64], bool G[32], bool D[32])`: Concaténation de Left et Right et exécution de la permutation inverse avec la table :

```
static int table_DES_IP_inv[64] = {
    40,8,48,16,56,24,64,32,
    39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30,
    37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28,
    35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26,
    33,1,41,9,49,17,57,25
};
```

- `chiffrement(bool inBloc[64], bool outBloc[64], bool key[64])`: Chiffrement d'un bloc de 64 bits en utilisant les fonctions : `fonctionIP`, `KS`, `fonctionF`, `XOR32`, `fonctionIP_inv`.

3. Créer un fichier « Main.cpp » qui contient le programme principal `int main ( int argc , char *argv[] )`

Le programme principal contient les étapes suivantes :

- Créer un objet `myObj` de la classe DES ;
- Préparer un bloc de données de 64 bits et une clé de chiffrement de 64 bits :

```
unsigned char keyBin[8] = {0xF3, 0x7B, 0xED, 0x36, 0xAA, 0x5C, 0x3F, 0x7E};
unsigned char inBlocBin[8] = {0xE1, 0x39, 0xA9, 0x06, 0x0A, 0x18, 0x17, 0x1E};
```

- Convertir les deux blocs en blocs de booléens en utilisant les méthodes :

```
binToBool64(keyBin, keyBool);
binToBool64(inBlocBin, inBlocBool);
```

- Appeler la méthode `chiffrement(inBlocBool, outBlocBool, keyBool);` de l'objet `myObj`.
- Pour vérifier le résultat, on appel la méthode `boolToBin64(outBlocBool, outBlocBin);` puis vérifier le contenu du bloc `outBlocBin` en utilisant la méthode d'affichage `afficheBinBloc8(outBlocBin);`