## Description du processus de déchiffrement DES

**Conversion binaire**

Avant de commencer le processus de déchiffrement, le message chiffré fourni devra être sous forme non-binaire puisque la conversion sera elle-même gérée par le programme. La clé, quant à elle, devra bien entendu être fournie sous forme binaire.

**Création de 16 sous-clés**

Le processus de déchiffrement débute donc de la même manière que pour le chiffrement, c’est-à-dire par l’éclatement de la clé d’origine en 16 clés différentes comme expliqué dans le cours fourni.

**Paquetage**

Pour exemple, nous allons reprendre le message chiffré obtenu dans la partie décrivant le processus de chiffrement DES du cours :

**M = 1000100000110110101000010001001111001011011000001001010010010000**

Une fois la conversion binaire du message effectuée, nous obtenons un message binaire d’une longueur de 64 bits. Une longueur d’un multiple de 64 bits est importante sans quoi le message ne pourra être déchiffré.

Après cela, nous pouvons diviser le message en paquets de 64 bits. Si un paquet n’est pas sur 64 bits, le dernier par exemple, alors on le complète par des bits de 0. Ces paquets seront nommés **M1**, **M2,** etc. comme dans le cours.

**Permutation initiale**

Tout comme pour le chiffrement, nous effectuons la même permutation initiale pour le déchiffrement. Par exemple, le résultat de cette permutation avec **M1** sera :

**PI[M1] = 0011000011001010010000100001110011010101001001100001000100011010**

**Gauche et droite**

On note G la partie gauche de **PI[M1]** correspondant aux 32 premiers bits et **D** la partie droite correspondant aux 32 derniers.

**G = 00110000110010100100001000011100 D = 11010101001001100001000100011010**

**Rondes**

Tout comme pour le chiffrage, il faut effectuer 16 rondes dans le **sens** **inverse**. De ce fait, la ronde **k** correspondra au morceau **K16-k**  qui sera utilisé. Le schéma des rondes est le suivant :

1. On applique la fonction d’expansion au bloc **G**. On obtient un message **E[G]** sur 48 bits, qui sera considéré comme 12 blocs de 4 bits. Notons ici que nous commençons par la partie gauche du message et non par la partie droite comme c’est le cas dans le chiffrement.
2. On calcule **E[G] ou exclusif K16-k** lors de la ronde **k** en additionnant (exclusivement) avec le morceau de clef.
3. On découpe ensuite **E[G] ou exclusif K16-k** en 8 blocs de 6 bits. Notons **B1, …, B8** ces 8 blocs en écriture binaire dont l’écriture sera **x1x2x3x4x5**. **n = x1x5** et est compris entre 0 et 3, et **m = x2x3x4x5** et est compris entre 0 et 15. On récupérera le nombre dans la matrice de substitution **Si** à la ligne **n+1** et à la colonne **m+1**, puis on remplacera **Bi** par ce nombre convertit en binaire.
4. L’application successive de ces matrices de substitution sera notée **S**. Le résultat de l’étape précédente sera donc nommé **S[E[G] ou exclusif K16-k].** On lui appliquera alors la permutation des rondes et le résultat sera noté **P[S[E[G] ou exclusif K16-k]]**.
5. On remplacera **G** par **P[S[E[G] ou exclusif K16-k]]** **ou exclusif D** et **D** par **G**

Réalisation de la première ronde :

1. **E[G] = 000110100001011001010100001000000100000011111000**
2. On réalise le ou exclusif avec la clé **K16** :

**000110100001011001010100001000000100000011111000 ou exclusif**

**101101010001001010001011001110110111001101101100 =**

**101011110000010011011111000110110011001110010100**

Ainsi **E[G] ou exclusif K16 = 101011110000010011011111000110110011001110010100**

1. On regarde **E[G] ou exclusif K16** comme 8 blocs de 6 bits.

**E[G] ou exclusif K16 = 101011 110000 010011 011111 000110 110011 001110 010100**

* 1. **On va remplacer le bloc 101011 à l’aide de S1.**

Ligne : x1x5 = 11 = 3 base 10

Colonne : x2x3x4x5 = 0101 = 5 base 10

A la ligne 3 + 1 et la colonne 5 + 1 de S1 on trouve 9, ou 1001

* 1. **On va remplacer le bloc 110000 à l’aide de S2.**

Ligne : x1x5 = 10 = 2 base 10

Colonne : x2x3x4x5 = 1000 = 8 base 10

A la ligne 2 + 1 et la colonne 8 + 1 de S2 on trouve 5, ou 0101

* 1. **On va remplacer le bloc 010011 à l’aide de S3.**

Ligne : x1x5 = 01 = 1 base 10

Colonne : x2x3x4x5 = 1001 = 9 base 10

A la ligne 1 + 1 et la colonne 9 + 1 de S3 on trouve 8, ou 1000

* 1. **On va remplacer le bloc 011111 à l’aide de S4.**

Ligne : x1x5 = 01 = 1 base 10

Colonne : x2x3x4x5 = 1111 = 15 base 10

A la ligne 1 + 1 et la colonne 15 + 1 de S4 on trouve 9, ou 1001

* 1. **On va remplacer le bloc 000110 à l’aide de S5.**

Ligne : x1x5 = 00 = 0 base 10

Colonne : x2x3x4x5 = 0011 = 3 base 10

A la ligne 0 + 1 et la colonne 3 + 1 de S5 on trouve 1, ou 0001

* 1. **On va remplacer le bloc 110011 à l’aide de S6.**

Ligne : x1x5 = 11 = 3 base 10

Colonne : x2x3x4x5 = 1001 = 9 base 10

A la ligne 3 + 1 et la colonne 9 + 1 de S6 on trouve 14, ou 1110

* 1. **On va remplacer le bloc 001110 à l’aide de S7.**

Ligne : x1x5 = 00 = 0 base 10

Colonne : x2x3x4x5 = 0111 = 7 base 10

A la ligne 0 + 1 et la colonne 7 + 1 de S7 on trouve 13, ou 1101

* 1. **On va remplacer le bloc 010100 à l’aide de S8.**

Ligne : x1x5 = 00 = 0 base 10

Colonne : x2x3x4x5 = 1010 = 10 base 10

A la ligne 0 + 1 et la colonne 10 + 1 de S8 on trouve 3, ou 0011

* 1. **En conclusion, nous obtenons le message suivant :**  **S[E[G] ou exclusif K16] = 1001 0101 1000 1001 0001 1110 1101 0011**

1. **P[S[E[G] ou exclusif K16]] = 10110010101100100100100101011011**
2. Pour finir on réalise l’opération **P[S[E[G] ou exclusif K16]] ou exclusif D** qui deviendra le nouveau **G** et le nouveau **D** sera l’ancien **G**

**10010101100010010001111011010011 ou exclusif**

**11010101001001100001000100011010 =**

**01100111100101000101100001000001**

Donc :

**G = 01100111100101000101100001000001 D = 11010101001001100001000100011010**

Voici le détail des résultats pour chaque fin de ronde :

1. **G=10011101001010010000000111010001 D=00110000110010100100001000011100**
2. **G=00101111100101010111011000111111 D=01100111100101000101100001000001**
3. **G=00010111101001110010010111110111 D=00101111100101010111011000111111**
4. **G=01000110000011010111100010111011 D=00010111101001110010010111110111**
5. **G=00101111001001110101000100100100 D=01000110000011010111100010111011**
6. **G=01100001011001111100011111101100 D=00101111001001110101000100100100**
7. **G=10110001110000011011010001001001 D=01100001011001111100011111101100**
8. **G=11011000000101101110110100111101 D=10110001110000011011010001001001**
9. **G=00110000100011100000011111011101 D=11011000000101101110110100111101**
10. **G=11011110001100011000100010010110 D=00110000100011100000011111011101**
11. **G=11000010011110110010001010100101 D=11011110001100011000100010010110**
12. **G=00010110110001011111000000000101 D=11000010011110110010001010100101**
13. **G=01101111000010101101000101000010 D=00010110110001011111000000000101**
14. **G=11011110111011001101000011001100 D=01101111000010101101000101000010**
15. **G=01111111101100100000001111110010 D=11011110111011001101000011001100**
16. **G=01111101101010110011110100101010 D=01111111101100100000001111110010**

On finira par concaténer les deux parties de la dernière ronde pour obtenir le message suivant :

**M’ = 0111110110101011001111010010101001111111101100100000001111110010**

**Permutation initiale inverse**

Tout comme pour le chiffrement, on applique la permutation initiale inverse à notre message pour le déchiffrer :

**PI[M’] = 1101110010111011110001001101010111100110111101111100001000110010**

Le premier paquet du message a été déchiffré. Il suffira donc d’appliquer le même processus à tous les autres paquets du message puis de les concaténer ensemble afin d’obtenir le message déchiffré.

Bien entendu, il faudra convertir le résultat binaire déchiffré afin de pouvoir le rendre lisible.