Projet de Cryptanalyse: Cryptanalyse sur le DES à 8 tours

Maxime Bineau, Nicolas Grellety, Bowen Liu

8 décembre 2017

1 Question 1 : Programmation du DES-8

DES est un chiffrement par bloc, c'est-à-dire qu'il fonctionne sur des blocs en clair d'une taille donnée (64 bits) et renvoie des blocs de texte chiffré de la même taille. Ainsi, le DES aboutit à une permutation entre les 2^{64} des arrangements possibles de 64 bits, chacun pouvant être soit 0 soit 1. Chaque bloc de 64 bits est divisé en deux blocs de 32 bits chacun, un demi-bloc gauche L et un demi-droite R. (Cette division n'est utilisée que dans certaines opérations.)

Soit M le message en texte brut où M est au format binaire, on obtient le bloc de texte 64 bits.

$$M = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001\ 1010\ 1011\ 1100\ 11011110\ 1111$$
$$L = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111$$
$$R = 1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111$$

Le premier bit de M est "0". Le dernier bit est "1". Nous lisons de gauche à droite.

DES fonctionne sur les blocs de 64 bits en utilisant des tailles de clé de 56 bits. Les clés sont en réalité stockées comme étant de longueur de 64 bits, mais tous les huitièmes bits de la clé ne sont pas utilisés (c'est-à-dire les bits numérotés 8, 16, 24, 32, 40, 48, 56 et 64). Cependant, nous allons néanmoins numéroter les bits de 1 à 64, en allant de gauche à droite, dans les calculs suivants. Mais, comme vous le verrez, les huit bits que nous venons de mentionner sont éliminés lorsque nous créons des sous-clés.

$$K = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 111100014$$

L'algorithme DES utilise les étapes suivantes :

Étape 1 : Créer 16 sous-clés, dont chacune a une longueur de 48 bits.

La clé de 64 bits est permutée selon le tableau suivant, PC-1. Puisque la première entrée dans la table est "57", cela signifie que le 57ème bit de la clé originale K devient le premier bit de la clé permutée K+. Le 49ème bit de la clé d'origine devient le deuxième bit de la clé permutée. Le 4ème bit de la clé d'origine est le dernier bit de la clé permutée. Notez que seulement 56 bits de la clé d'origine apparaissent dans la clé permutée.

```
PC1 = [  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  ]
```

Exemple : À partir de la clé 64 bits d'origine

$$K = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001$$

nous obtenons la permutation de 56 bits suivante

$$K+ = 1111000\ 0110011\ 0010101\ 0101111\ 0101010\ 1011001\ 1001111\ 0001111$$

Ensuite, on divise cette clé en une moitié gauche et une moitié droite, C0 et D0, où chaque moitié a 28 bits.

Exemple : A partir de la clé permutée K+, on obtient

$$C_0 = 1111000\ 0110011\ 0010101\ 0101111$$
$$D_0 = 0101010\ 1011001\ 1001111\ 0001111$$

Avec C_0 et D_0 définis, on crée maintenant seize blocs C_n et D_n, 1 <= n <= 8. Chaque paire de blocs C_n et D_n est formée à partir de la paire précédente C_{n-1} et D_{n-1}, respectivement, pour n = 1, 2, ..., 8, en utilisant la programmation suivante des "décalages à gauche" du bloc précédent. Pour faire un décalage vers la gauche, on déplace chaque bit d'une place vers la gauche, sauf pour le premier bit, qui est cyclé jusqu'à la fin du bloc.

Iteration numéro	Nombre de décalage à gauche
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2

$$Shift\_Dis = [1, 1, 2, 2, 2, 2, 2, 2]$$

Cela signifie, par exemple, que C_3 et D_3 sont obtenus à partir de C_2 et D_2, respectivement, par deux décalages à gauche, et C_8 et D_8 sont obtenus à partir de C_7 et D_7, respectivement, par un décalage à gauche.

Exemple : A partir de la paire de paires originales C_0 et D_0, on obtient :

$$C_0 = 11110000110011001010101111$$
$$D_0 = 01010101011001100111100011111$$
$$C_1 = 11100001100110010101011111$$
$$D_1 = 10101010110011001111100011110$$
$$C_2 = 11000011001100101010111111$$
$$D_2 = 01010101100110011111000111101$$
$$C_3 = 00001100110010101010111111111$$
$$D_3 = 01010110011001111100011110101$$
$$C_4 = 00110011001010101011111111100$$
$$D_4 = 01011001100111110001111010101$$
$$C_5 = 11001100101010101111111110000$$
$$D_5 = 01100110011110000111101010101$$
$$C_6 = 00110010101010111111111000011$$
$$D_6 = 10011001111000111101010101011$$
$$C_7 = 11001010101011111111100001100$$
$$D_7 = 01100111100011110101010101110$$
$$C_8 = 00101010101111111110000110011$$
$$D_8 = 10011110001111010101011001$$

Nous formons maintenant les clés K_n, pour $1 \leqslant n \leqslant 8$, en appliquant la table de permutation suivante à chacune des paires concaténées C_nD_n. Chaque paire a 56 bits, mais PC-2 n'en utilise que 48.

PC2= [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]

Par conséquent, le premier bit de K_n est le 14ème bit de CnDn, le second bit le 17ème, et ainsi de suite, se terminant par le 48ème bit de K_n étant le 32ème bit de C_nD_n.

Exemple : Pour la première clé, nous avons

$$C_1D_1 = 1110000\ 1100110\ 0101010\ 1011111\ 1010101\ 0110011\ 0011110\ 0011110$$

qui, après avoir appliqué la permutation PC-2, devient

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

Pour les autres clés nous avons

$$K_2 = 011110\ 011010\ 111011\ 011001\ 110110\ 111100\ 100111\ 100101$$
$$K_3 = 010101\ 011111\ 110010\ 001010\ 010000\ 101100\ 111110\ 011001$$
$$K_4 = 011100\ 101010\ 110111\ 010110\ 110110\ 110011\ 010100\ 011101$$
$$K_5 = 011111\ 001110\ 110000\ 000111\ 111010\ 110101\ 001110\ 101000$$
$$K_6 = 011000\ 111010\ 010100\ 111110\ 010100\ 000111\ 101100\ 101111$$
$$K_7 = 111011\ 001000\ 010010\ 110111\ 111101\ 100001\ 100010\ 111100$$
$$K_8 = 111101\ 111000\ 101000\ 111010\ 110000\ 010011\ 101111\ 111011$$

Voilà pour les sous-clés. Maintenant, nous regardons le message lui-même.

Étape 2 : On encode chaque bloc de données de 64 bits.

On divise le bloc permuté IP en une moitié gauche L_0 de 32 bits, et une moitié droite R_0 de 32 bits.

Exemple : De IP, on obtient L_0 et R_0

$$L_0 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$
$$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

On passe maintenant par 16 itérations, pour $1 \leqslant n \leqslant 16$, en utilisant une fonction f qui opère sur deux blocs - un bloc de données de 32 bits et une clé K_n de 48 bits - pour produire un bloc de 32 bits. Soit + l'addition XOR, (addition bit à bit modulo 2). Alors pour n allant de 1 à 16 nous calculons

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

Ceci conduit à un bloc final, pour n = 8, de L_8R_8. C'est-à-dire qu'à chaque itération, nous prenons les 32 bits de droite du résultat précédent et en faisons les 32 bits de gauche du pas courant. Pour les 32 bits de droite dans l'étape courante, on XOR les 32 bits de gauche de l'étape précédente avec le calcul f.

Exemple : Pour n = 1, nous avons

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$
$$L_1 = R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$
$$R_1 = L_0 + f(R_0, K_1)$$

Il reste à expliquer comment fonctionne la fonction f. Pour calculer f, nous étendons d'abord chaque bloc R_{n-1} de 32 bits à 48 bits. Ceci est fait en utilisant une table de sélection qui répète certains des bits de R_{n-1}. Nous appellerons l'utilisation de cette table de sélection la fonction E. Ainsi $E(R_{n-1})$ a un bloc d'entrée de 32 bits, et un bloc de sortie de 48 bits.

Soit E tel que les 48 bits de sa sortie, écrits en 8 blocs de 6 bits chacun, soient obtenus en sélectionnant les bits dans ses entrées dans l'ordre suivant le tableau suivant :

$$
\begin{array}{llllllll}
\mathrm{E} = [& 32, & 1, & 2, & 3, & 4, & 5, & \\
& 4, & 5, & 6, & 7, & 8, & 9, & \\
& 8, & 9, & 10, & 11, & 12, & 13, & \\
& 12, & 13, & 14, & 15, & 16, & 17, & \\
& 16, & 17, & 18, & 19, & 20, & 21, & \\
& 20, & 21, & 22, & 23, & 24, & 25, & \\
& 24, & 25, & 26, & 27, & 28, & 29, & \\
& 28, & 29, & 30, & 31, & 32, & 1 &]
\end{array}
$$

Ainsi, les trois premiers bits de $E(R_{n-1})$ sont les bits dans les positions 32, 1 et 2 de R_{n-1} tandis que les deux derniers bits de $E(R_{n-1})$ sont les bits dans les positions 32 et 1.

Exemple : Nous calculons $E(R_0)$ à partir de R_0 comme suit :

$$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

$$E(R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$$

(Note : Chaque bloc de 4 bits d'origine a été étendu à un bloc de 6 bits de sortie.)

Ensuite, dans le calcul de f, on XOR la sortie $E(R_{n-1})$ avec la clé K_n :

$$K_n + E(R_{n-1})$$

Par exemple : Pour K_1, $E(R_0)$, nous avons

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

$$E(R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$$

$$K_1 + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111$$

.

Nous n'avons pas encore fini de calculer la fonction f. À ce stade, nous avons étendu R_{n-1} de 32 bits à 48 bits, en utilisant la table de sélection, et XOR le résultat avec la clé K_n. Nous avons maintenant 48 bits, ou huit groupes de six bits. Nous faisons maintenant quelque chose d'étrange avec chaque groupe de six bits : nous les utilisons comme des adresses dans des tables appelées "S Box". Chaque groupe de six bits nous donnera une adresse dans une boîte S différente. Situé à cette adresse sera un nombre de 4 bits. Ce nombre de 4 bits remplacera les 6 bits d'origine. Le résultat net est que les huit groupes de 6 bits sont transformés en huit groupes de 4 bits (les sorties à 4 bits des boîtes S) pour un total de 32 bits.

On écrit le résultat précédent, qui est 48 bits, sous la forme :

$$K_n + E(R_{n-1}) = B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ B_8$$

où chaque B_i est un groupe de six bits. Nous calculons maintenant

$$S_1(B_1)\ S_2(B_2)\ S_3(B_3)\ S_4(B_4)\ S_5(B_5)\ S_6({}_B6)\ S_7(B_7)\ S_8(B_8)$$

où Si (B_i) se réfère à la sortie de la i-ème boîte.

Pour répéter, chacune des fonctions S_1, S_2, ..., S_8 prend un bloc de 6 bits en entrée et fournit un bloc de 4 bits en sortie. Le tableau pour déterminer S_1 est montré et expliqué ci-dessous :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Si S_1 est la fonction définie dans ce tableau et B est un bloc de 6 bits, alors S_1 (B) est déterminé comme suit : Le premier et le dernier bits de B représentent dans la base 2 un nombre dans la gamme décimale 0 à 3 (ou binaire 00 à 11). Que ce nombre soit i. Les 4 bits moyens de B représentent dans la base 2 un nombre dans la gamme décimale 0 à 15 (binaire 0000 à 1111). Que ce nombre soit j. Recherchez dans le tableau le nombre de la ligne i et de la colonne. C'est un nombre compris entre 0 et 15 et est uniquement représenté par un bloc de 4 bits. Ce bloc est la sortie S_1 (B) de S1 pour l'entrée B. Par exemple, pour le bloc d'entrée B = 011011, le premier bit est "0" et le dernier bit "1" donne 01 comme ligne. C'est la ligne 1. Les quatre bits du milieu sont "1101". C'est l'équivalent binaire de décimal 13, donc la colonne est la colonne numéro 13. Dans la ligne 1, la colonne 13 apparaît 5. Cela détermine la sortie ; 5 est binaire 0101, de sorte que la sortie est 0101. D'où S_1 (011011) = 0101.

Les tables définissant les fonctions $S_1, \ldots, S_8$ sont les suivantes :

```
SBOX = []
SBOX.append([ 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  ])

SBOX.append([ 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  ])

SBOX.append([ 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  ])

SBOX.append([  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  ])

SBOX.append([  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  ])

SBOX.append([ 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  ])

SBOX.append([  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  ])

SBOX.append([ 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  ])
```

Exemple : Pour le premier tour, on obtient comme sortie des huit cases S :

$$K_1 + E(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$$

$$S_1(B_1)\ S_2(B_2)\ S_3(B_3)\ S_4(B_4)\ S_5(B_5)\ S_6(B_6)\ S_7(B_7)\ S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

La dernière étape du calcul de f est de faire une permutation P de la sortie S-box pour obtenir la valeur finale de f :

$$f = P((S_1(B_1)\ S_2(B_2)\ S_3(B_3)\ S_4(B_4)\ S_5(B_5)\ S_6(B_6)\ S_7(B_7)\ S_8(B_8))$$

La permutation P est définie dans le tableau suivant. P génère une sortie 32 bits à partir d'une entrée 32 bits en permutant les bits du bloc d'entrée.

$$\begin{array}{llllllll} \text{P} = [& 16, & 7, & 20, & 21, & 29, & 12, & 28, & 17, \\ & 1, & 15, & 23, & 26, & 5, & 18, & 31, & 10, \\ & 2, & 8, & 24, & 14, & 32, & 27, & 3, & 9, \\ & 19, & 13, & 30, & 6, & 22, & 11, & 4, & 25 \quad] \end{array}$$

Exemple : De la sortie des huit S-Box :

$$S_1(B_1)\ S_2(B_2)\ S_3(B_3)\ S_4(B_4)\ S_5(B_5)\ S_6(B_6)\ S_7(B_7)\ S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$$

on a

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

Donc,

$$R_1 = L_0 + f(R_0, K_1)$$

$$R1 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111 \oplus 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$R_1 = 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100$$

Au tour suivant, nous aurons $L_2 = R_1$, qui est le bloc que nous venons de calculer, puis nous devons calculer $R_2 = L_1 + f(R_1, K_2)$, et ainsi de suite pendant 8 tours. A la fin du seizième tour, nous avons les blocs L_8 et R_8. Nous inversons ensuite l'ordre des deux blocs dans le bloc de 64 bits

$$R_8 L_8$$

Exemple : Si nous traitons tous les 16 blocs en utilisant la méthode définie précédemment, nous obtenons, au 8ème tour,

$$L_8 = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$$

$$R_8 = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$$

Nous inversons l'ordre de ces deux blocs.

2 Question 2 : Matrice des approximations linéaires

La matrice des approximations linéaires de la SBox S_5

$L[\alpha, \beta] = \text{Card}\ \{x \in F_2^6, \langle \alpha, x \rangle + \langle \beta, S_5(x) \rangle = 0\}$ est disponible en lançant la fonction *Card_ L* présente dans notre fichier principal.

Les lignes qui suivent dans le code permettent d'afficher correctement la matrice.

3 Question 3 : Propriétés de la fonction f

On cherche la probabilité d'avoir $X[16] = Y[2] + Y[7] + Y[13] + Y[24]$, avec X chaîne de 32 bits, et $Y = f(X, K)$, K étant fixé.

En partant d'un message X de 32 bits, on va devoir l'étendre, quand on regarde $X[16]$ (17 sur le papier du NIST, car leurs indices commencent à 1), on voit qu'il sera présent à 2 endroits lors de l'extension, sur le dernier bit de la quatrième SBox, et sur le deuxième de la SBox 5.

K étant fixé le XOR ne va pas intervenir, on s'intéresse donc maintenant aux boîtes S.

En regardant la SBox 5 (on ne regardera pas la quatrième), notre bit $X[16]$ va agir sur les bits 17, 18 ,19 ,20 (selon la notation du NIST) du message avant la permutation, et en regardant ce qu'il se passe après permutation, on voit que nos bits sont maintenant aux positions 2, 7, 13 ,24 (avec la notation d'indice habituelle), et donc cela correspond à la probabilité que l'on cherche.

Pour calculer la probabilité de cet événement, on va utiliser le Piling Up Lemma, et de la matrice L calculée plus tôt.

On regarde alors notre matrice avec $\alpha = 16, \beta = 15$, et on va trouver que la probabilité que $X[16] = Y[2] + Y[7] + Y[13] + Y[24]$ est de $0.5 \pm \frac{20}{64}$.

4 Question 4 : Propriétés après le 3e tour

On cherche à connaître la probabilité de l'équation suivante :

$$L_0[2] + L_0[7] + L_0[13] + L_0[24] + R_0[16] + R_3[2] + R_3[7] + R_3[13] + R_3[24] + L_3[16] = 0$$

En s'appuyant de la question précédente, on peut poser

$$R_0[16] = Y_0[2] + Y_0[7] + Y_0[13] + Y_0[24], avec Y_0 = f(R_0, K)$$

or on sait que

$$Y_0[2] + L_0[2] = R_1[2]$$

de même pour les autres indices. On a alors

$$R_0[16] + L_0[2] + L_0[7] + L_0[13] + L_0[24] = R_1[2] + R_1[7] + R_1[13] + R_1[24]$$

on a la même chose pour 3 :

$$R_2[16] + L_2[2] + L_2[7] + L_2[13] + L_2[24] = R_3[2] + R_3[7] + R_3[13] + R_3[24]$$

mais

$$R_2[16] = L_3[16], L_2 = R_1$$

donc en additionnant les deux équations, on trouve l'équation que l'on cherche.

La probabilité associée à cette équation résulte du Piling Up Lemma cette fois-ci en prenant compte les deux probabilités. On obtient donc $0.5 \pm 2 * \frac{20^2}{64^2}$

5 Question 5 : DES-8 avec la même clef, quatrième tour

On prendra garde à la notation des indices de 1 à n sur l'ensemble de la réponse.

Soient $\alpha, \beta \in 0, 1$ deux fautes. Il nous est demandé de suivre ces derniers jusqu'au tour 4 afin de déterminer les indices ne différant pas entre L_4 et $L_4^\star$ ainsi qu'avec R_4 et $R_4^\star$.

Avec le premier tour qui nous est donnée, nous avons :
$R_1 = R_1^\star$, $L_1 \oplus L_1^\star = 0\alpha\beta 0000000000000000000000000000$
Donc $R_1 \oplus R_1^\star = 0 \cdots 0$

Au deuxième tour, nous posons alors :
$L_2 \oplus L_2^\star = R_1 \oplus R_1^\star = 0 \cdots 0$
$R_2 \oplus R_2^\star = L_1 \oplus L_1^\star \oplus f(R_1, K_2) \oplus f(R_1^\star, K_2)$
Or, $R_1 \oplus R_1^\star = 0 \cdots 0$
donc, $R_2 \oplus R_2^\star = L_1 \oplus L_1^\star = 0\alpha\beta 0 \cdots 0$

À partir du troisième tour, $R_2 \oplus R_2^\star \neq 0$. Nous avons alors :
$L_3 \oplus L_3^\star = 0\alpha\beta 0 \cdots 0$,
$R_3 \oplus R_3^\star = L_2 \oplus L_2^\star \oplus f(R_2, K_3) \oplus f(R_2^\star, K_3) = f(R_2, K_3) \oplus f(R_2^\star, K_3)$

Comme indiqué, $R_2 \oplus R_2^\star \neq 0$. En passant dans la fonction f, nous avons alors :
• Par la fonction E :
l'indice 2 va en 3^{ime} positon
l'indice 3 va en 4^{ime} position

On rentre ensuite dans la première S-Box et on modifie donc les quatre premiers indices.

• Enfin, par la fonction P :
l'indice 1 va en 9^{ime} position
l'indice 2 va en 17^{ime} position
l'indice 3 va en 23^{ime} position
l'indice 4 va en 31^{ime} position

On a donc $R_3 \oplus R_3^\star = 0 \cdots 0 * 0 \cdots 0 * 0 \cdots 0 * 0 \cdots 0 * 0$

Pour finir, au quatrième tour nous avons :
$L_4 \oplus L_4^\star = R_3 \oplus R_3^\star = 0 \cdots 0 * 0 \cdots 0 * 0 \cdots 0 * 0 \cdots 0 * 0$

$R_4 \oplus R_4^\star = L_3 \oplus L_3^\star \oplus f(R_3, K_4) \oplus f(R_3^\star, K_4) = 0\alpha\beta 0 \cdots 0 \oplus f(R_3, K_4) \oplus f(R_3^\star, K_4)$

• Par la fonction E :
l'indice 9 va en 12^{ime} position et 14^{ime} position
l'indice 17 va en 24^{ime} position et 26^{ime} position
l'indice 23 va en 34^{ime} position
l'indice 31 va en 46^{ime} position

On rentre donc dans les S-Box 2, 3, 4, 5, 6 et 8. Seuls les indices des S-Box 1 et 7 ne seront pas modifiés.

• Par la fonction P :

Départ	Arrivée	Départ	Arrivée
5	13	16	1
6	28	17	8
7	2	18	14
8	18	19	25
9	24	20	3
10	16	21	4
11	30	22	29
12	6	23	11
13	26	24	19
14	20	29	5
15	10	30	27
31	15	32	21

Nous avons donc $R_4 \oplus R_4^\star = 0\alpha\beta 0 \cdots 0 \oplus f(R_3, K_4) \oplus f(R_3^\star, K_4) = \star \cdots \star\, 0 \star 0 \star \star 0 \star \cdots \star\, 0 \star \cdots \star\, 00 \star \cdots \star\, 00$

En conclusion, pour $L_4 \oplus L_4^\star$, les indices des S-Box de 2 à 8 ne diffèrent pas et pour $R_4 \oplus R_4^\star$, seuls les indices des S-Box 1 et 7 ne diffèrent pas.

6 Question 6 : DES-8 avec la même clef, septième tour

En appliquant l'équation trouvée un peu plus haut pour le septième tour, on a

$$L_4[2] + L_4[7] + L_4[13] + L_4[24] + R_4[16] + R_7[2] + R_7[7] + R_7[13] + R_7[24] + L_7[16] = 0$$

et

$$L_4^*[2] + L_4^*[7] + L_4^*[13] + L_4^*[24] + R_4^*[16] + R_7^*[2] + R_7^*[7] + R_7^*[13] + R_7^*[24] + L_7^*[16] = 0$$

En additionnant les deux équations du dessus, on les L_4 n'étant pas différents aux indices 2,7,13,24 et les R_4 ne l'étant pas à l'indice 16 non plus, ils vont s'annuler et on va se ramener à notre équation cherchée :

$$R_7[2] + R_7[7] + R_7[13] + R_7[24] + L_7[16] + R_7^*[2] + R_7^*[7] + R_7^*[13] + R_7^*[24] + L_7^*[16] = 0$$

avec une probabilité d'environ 0,575

7 Question 7 : Récupération de bits de la dernière clef

Sachant que l'on connaît c = $L_8||R_8$ et c* = $L_8^*||R_8^*$, on va essayer de se ramener à l'équation de la question précédente, et de calculer cette probabilité.

On connaît déjà les R_7 car $L_8 = R_7$, il nous reste simplement à retrouver $L_7[16], L_7^*[16]$.

Or on sait que $L_7 = f(R_7, K) + R_8$, et en particulier $L_7[16]$ provient après permutation de la première SBox, et donc si on bruteforce les 6 premiers bits de K, on va pouvoir récupérer des potentiels $L_7[16]$ et tester l'équation précédente et vérifier que la probabilité pour cette clé est proche de celle que l'on cherche.

En pratique on constate que seulement la bonne clé aura une probabilité proche de celle théorique, et que les autres ont une probabilité bien plus proche de 0.5.

En utilisant la fonction *guess_bf_K8* dans notre fichier, on trouve que les 6 premiers bits de K sont [0, 1, 0, 0, 1, 0]

8 Question 8 : Construction des couples (c, c*)

En connaissant les 6 premiers bits de K_1, on va pouvoir prédire la sortie de la première SBox afin d'influer sur les bits qui en sortie de f seront les bits 8, 16, 22, 30.

On a alors juste à construire les bits 8, 16, 22, 30 de m à partir de notre calcul, et de cette façon, les modifications introduites dans la partie gauche, lors du xor, vont s'annuler afin de retrouver seulement des modifications aux indices 1 et 2 après le premier tour.

La fonction *gen_ M_ change* permet de générer notre ensemble M de message, la fonction *one_ tour*, de faire notre tour de DES, et donc nous construisons ensuite dans notre fonction *gen_ couple_ M*, l'ensemble de couples (c, c*)

9 Question 9 : Attaque à clairs choisis

Dans cette partie, on souhaite faire une attaque à clair choisis.

On choisit donc les couples de messages de la question précédente. En utilisant 1 seul message, on arrive à créer 96 couples (c, c*) avec les bonnes propriétés.

Néanmoins, 96 couples ne sont pas suffisants, on génère donc 10 messages auxquels on ré applique la question 8. Cela va donc nous donner 10 * 96 couples de messages.

On passe tous ces couples de messages à la fonction DES-8 avec la même clef initiale.

La phase d'attaque :

On va faire une attaque bruteforce sur la clé K_1. Pour chacun de ces essais, nous allons appliquer la fonction DES-8 aux 960 couples générés.

Cela nous donne pour chaque appels de la fonction 960 couples de chiffrés.

A chaque paire de 960 couples de chiffrés, on applique la question 6 afin d'obtenir la probabilité associée ainsi que les 6 premiers bits de la clef K8 associée.

A la fin, en choisissant la plus grande probabilité, on retrouve les 6 premiers bits des clefs K_1 et K_8.

10 Question 10 : Nouvelle idée

Par la question 2, lorsqu'on affiche la matrice d'approximation linéaire L, on peut constater que des valeurs s'éloignent de manière plus ou moins importante de la valeur 32. Nous pouvons par exemple retrouver à l'intérieur de cette dernière les valeurs 12, 16 et 18.

De cette manière, nous avons des valeurs qui modifient les propriétés de la fonction F. Ces valeurs correspondent à des positions faibles qui, au final, peuvent nous permettre de retrouver des bits de clef.