Cryptographie et Sécurité
Série 6 : TP Python - DES

27 Octobre 2022

A rendre sur **Moodle**, sous forme de fichier **.py**, avant le **Mercredi 2 Novembre 2022 à 23h59**.

Votre code doit être **suffisamment commenté**.

Implémentation de DES

Le but de cette série est "tout simplement" d'implémenter DES (Encryption et Decryption).

Tables de Permutation

DES nécessite de nombreuses tables de permutation, expansion et substitution de bits. Le nom de ces tables est indiqué en **gras** dans les explications ci-après.

Une table se lit de la manière suivante :

7	10	4	1
6	2	8	12
11	3	5	9

Signifie que le 7ème bit passe en première place, le 10ème passe en 2ème place, etc.
Donc si l'entrée était "abcdefghijkl", la sortie sera "gjdafbhlkcei" (évidemment, l'entrée sera en binaire et pas en caractères, mais il s'agit uniquement d'un exemple illustratif).

Toutes les tables sont déjà dans le fichier Python "DES_tables.py" fourni sur Moodle !

Elles y sont sous forme de listes. Par exemple, la table montrée comme exemple ci-dessus serait exprimée comme la liste de 12 éléments : [7,10,4,1,6,2,8,12,11,3,5,9].

Toutes les tables sont sous la forme d'une seule liste comprenant toutes les valeurs, excepté les S-Boxes, qui de par leur fonctionnement, sont exprimées comme une liste contenant 4 sous-listes (pour les lignes) de 16 valeurs chacune (les colonnes). On explique bien sûr les principes de toutes ces boites plus loin.

Les tables sont également incluses dans ce PDF pour illustrer les différentes étapes.

Principe de DES

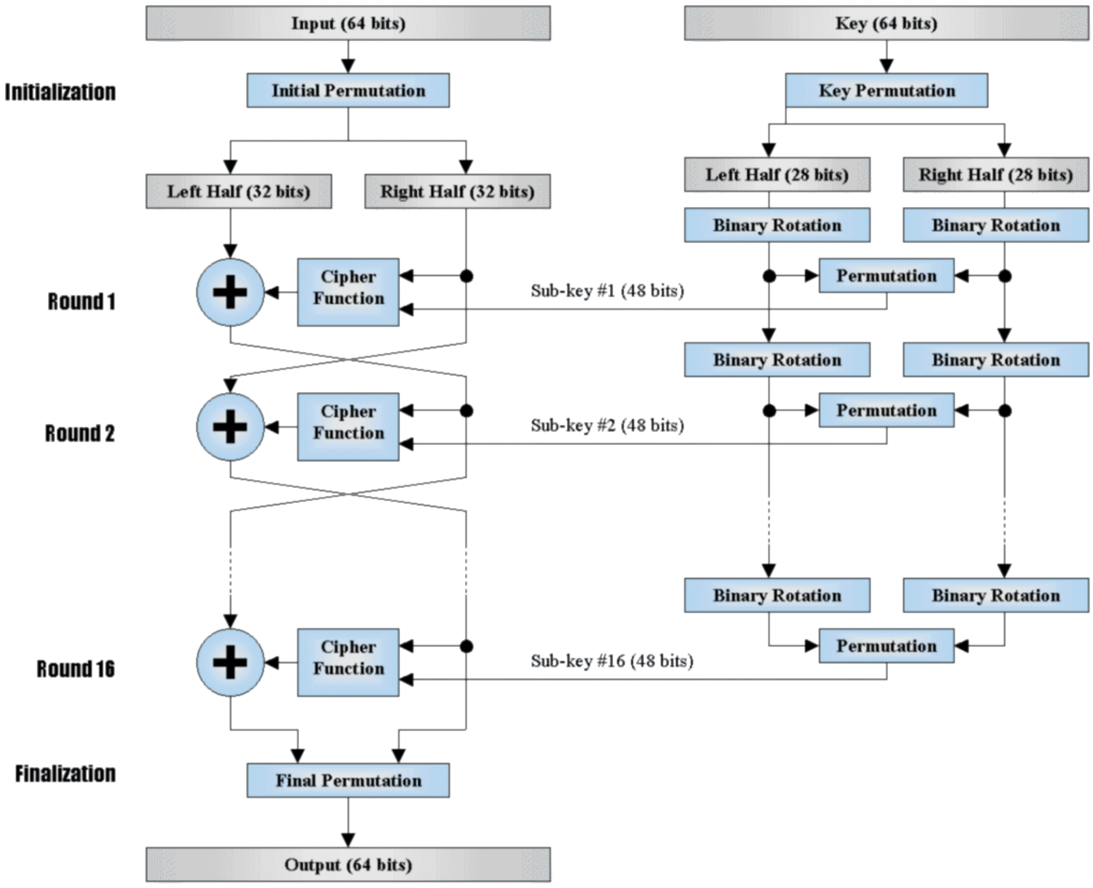

Figure 1: Schéma du fonctionnement de DES

- Le message fait 64 bits, la clé 56 bits effectifs, 64 au total avec 8 bits de parité.
- On applique au message une permutation initiale **IP**.

- Le message est ensuite scindé en deux parties de 32 bits, L_0 et R_0.
  A chaque étape, on a $L_{i+1} = R_i$ et $R_{i+1} = L_i \oplus Cipher(R_i, SubKey_i)$. On effectue 16 rounds ainsi.

- La cipher function prend une sous-clé de 48 bits et un R_i de 32 bits, et calcule un cipher L_{i+1} de 32 bits. On décrira la cipher function plus loin.

- La clé est séparée en deux moitiés de 28 bits , LK_0 et RK_0, par la permutation initiale **PC-1** (qui retire les 8 bits de parité par la même occasion).
  A chaque étape, on effectue une **rotation** de LK_i pour obtenir LK_{i+1}, et pareil pour RK_i.
  La sous-clé $SubKey_{i+1}$ est alors générée par une table de permutation **PC-2** appliquée à (LK_{i+1}, RK_{i+1}) (on a donc 56 bits, qu'on permute, puis on en extrait une sous-clé de 48 bits).
  Attention, LK_{i+1} et RK_{i+1} ne sont pas modifiés par cette permutation (on génère uniquement $SubKey_i$ sans les modifier).

- Enfin, lors de la 16ème étape, on inverse pas L et R, et on effectue une dernière permutation (qui est l'inverse de la permutation initiale) $\mathbf{FP} = \mathbf{IP}^{-1}$ pour obtenir le Ciphertext final.

- Pour la Décryption, c'est très facile, une fois que l'Encryption fonctionne, il suffit de réutiliser le même algorithme, en changeant uniquement l'ordre des 16 sous-clés. Il suffit donc de calculer les mêmes 16 sous-clés, d'inverser leur ordre (la 16ème devient la 1ère, ... , la 1ère devient la 16ème). Et ensuite on applique l'algorithme comme si c'était l'encryption.

Cipher Function

Il ne nous manque désormais que la Cipher Function. La cipher function consiste en 4 étapes :

1. On prend la partie R_i de 32 bits, et on l'étend sur 48 bits à l'aide de la table **E**. Elle fonctionne comme une table de permutation, à ceci près que certains bits seront répétés.
2. On effectue un XOR entre ce R_i étendu et la sous-clé de l'étape.
3. On découpe ce résultat en 8 blocs de 6 bits. Chaque bloc de 6 bits est réduit à un bloc de 4 bits en utilisant une des **8 S-boxes** S_1**,** S_2**, ...,** S_8.
   La première pour les 6 premiers bits, la deuxième pour les 6 suivants, etc...
   Une S box fonctionne comme suit :

   - Soit l'entrée de 6 bits $e = x_1x_2x_3x_4x_5x_6$ (donc avec x_1 le bit de poids le plus fort, x_6 le plus faible).

- On a une table de 16 colonnes et 4 lignes, numérotées de 0 à 15 et 0 à 3 respectivement.
  La valeur de sortie, sur 4 bits, donc une valeur entre 0 et 15, est indiquée par une colonne et une ligne.
- La ligne est donnée en binaire par x_1x_6 (par exemple, $x_1 = 0$ et $x_6 = 0$ indique la ligne $(00)_2$, donc la ligne 0. Avec $x_1 = 1$ et $x_6 = 0$, on a la ligne $(10)_2$, donc la ligne 2).
- La colonne est indiquée similairement par les bits $x_2x_3x_4x_5$ (par exemple, "0000" indique la colonne 0, "1001" indique la colonne 9, ...)

On repasse donc de 48 bits (8 fois 6 bits) à 32 bits (8 fois 4 bits).

4. Enfin, on applique aux 32 bits obtenus une permutation **P**.

On a donc $Cipher(R_i, SubKey_i) = P(S(E(R_i) \oplus SubKey_i))$.

Tables et Valeurs pour générer les clés

On a besoin de 3 choses :

- La table de permutation initiale **PC-1** (C donne la moitié Gauche, et D donne la moitié Droite), ainsi que la table pour la création des 16 sous-clés (la même table est utilisée à chaque étape) **PC-2** :

C						
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
D						
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

PC1

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

PC2

Figure 2: tables de permutations pour générer les clés

- Et les **rotations** binaires à chaque étape, qui sont très simples :
  On décale vers la gauche;
  De **1 bit** pour les étapes 1,2,9 et 16;
  Et de **2 bits** pour les autres étapes.

Tables de permutation initiale et finale

Les tables **IP** et $\mathbf{FP} = \mathbf{IP}^{-1}$:

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

Figure 3: Tables de permutation initiale et finale du message

Tables de la Cipher Function

Voici E, P et les S-Boxes de DES (toutes ces tables sont déjà incluses dans "DES_tables.py") :

DES: Expansion Permutation (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

Figure 4: Table d'Expansion E

P Box

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

Figure 5: Table de permutation P

S-Box

		Column															
Box	**Row**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
								S_1									
	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
								S_2									
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
								S_3									
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
								S_4									
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
								S_5									
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
								S_6									
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
								S_7									
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
								S_8									
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure 6: Les S-Box S_1 à S_8

Conseils d'implémentation

- Les tables de permutation contiennent des valeurs de 1 à n. Faites attention à cela lorsque vous les utilisez, car les indices de listes python démarrent

à 0.

- Souvenez vous que la dernière étape de calcul est particlière car on inverse pas les deux moitiés du message crypté avant la permutation finale.
- Rappelez-vous que lors d'une affectation directe de type "liste-a = liste-b", les deux listes pointent sur le même objet, et qu'une modification de l'une modifie également l'autre. Pour réellement séparer les deux, utilisez liste-a = liste-b.copy() (ou utilisez deepcopy pour des listes récursives). De la même manière, une liste passée en paramètre à une fonction sera modifiée si le paramètre est modifié à l'intérieur de la fonction.
- Découpez bien votre travail : Commencez par faire des fonctions pour les opérations de base nécessaire (par ex appliquer une permutation). Construisez séparément le bloc de génération de clés. De même, construisez séparément la cipher function.
- Testez chaque petite partie séparément pour vérifier leur bon fonctionnement.
- Lorsque votre code est terminé, vérifiez si vous obtenez les bons résultats : par exemple, le message $(0011223344556677)_{16}$, avec la clé $(0123456789ABCDEF)_{16}$, doit s'encrypter en $(CADB6782EE2B4823)_{16}$ (ici tout est exprimé en hexadecimal).
  Vous pouvez utiliser `https://gchq.github.io/CyberChef/#recipe=DES_Encrypt(%7B'option':'Hex','string':'0123456789abcdef'%7D,%7B'option':'Hex','string':''%7D,'ECB','Hex','Hex')&input=MDAxMTIyMzM0NDU1NjY3Nw` pour tester d'autres exemples (l'exemple donné ci-dessus est déjà mis en place sur cette page).
  Pensez ensuite à vérifier que votre décryption fonctionne en regardant si vous retombez bien sur le même message après avoir appliqué l'encryption puis la décryption (la decryption, c'est simplement appliquer l'encryption en inversant l'ordre des sous-clés). L'exemple est fourni dans le fichier python, avec la manière de le transformer facilement d'une écriture hexadécimale en une liste de 64 bits (cela facilite les tests, à moins que vous ne souhaitiez taper la liste de 64 bits à la main).
- Enfin, on suggère ici des listes de bits, mais vous pouvez utiliser d'autres types de structures (strings, entiers, etc) si cela vous parait plus simple.