Data Encryption Standard

Table des matières

De nos jours, nous envoyons énormément de données via internet et ces données peuvent être interceptées. Si elles ne sont pas chiffrées, il est alors possible de lire ce que l'on envoie sur le réseau. La cryptographie devient alors primordiale pour protéger nos données envoyées.

1 Notions de cryptographie

1.1 Présentation du DES

1.1.1 Historique

En 1973, le *National Bureau of Standards* demande un algorithme de chiffrement qui soit accessible à tous afin que tout le monde puisse avoir accès à un système de protection de données. Ainsi, est créé le DES. Il est maintenant obsolète car la clef de 56 bits est trop petite et un message crypté peut être déchiffré par une attaque par force brute. Cependant, le triple DES peut être utilisé plus sûrement mais est plus long à s'exécuter.
Aucune attaque, hormis une attaque par force brute, n'a permis de faire craquer cet algorithme.

1.1.2 Caractéristiques

Le DES est un algorithme de cryptage en bloc par bloc. C'est-à-dire que le fichier à crypter est découpé par blocs de 64 bits, pour que l'aglgorythme chiffre succéssivement les blocs du fichier à crypter. Il faut par la suite mettre bout à bout tout les blocs pour reconstituer le fichier chiffré.

De plus, le DES est un algorithme de cryptage dit symétrique : il utilise la même clef pour crypter et décrypter un fichier.

Les blocs que crypte l'algorithme font une taille de 64 bits. La clef utilisée fait 56 bits.

1.2 Actions possibles sur les blocs

1.2.1 XOR

Le "ou exclusif" noté XOR est une opération effectuée sur les bits dont la table est la suivante :

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Table du XOR

Le XOR est symbolisé par  dans les schémas.

Exemple :
A = [0, 1, 0, 1]
B = [1, 1, 0, 0]
A XOR B = [1, 0, 0, 1]

1.2.2 Permutation (P-box)

Une table de permutation permet de changer l'ordre d'un bloc de bits.

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

Exemple d'une P-box utilisée par le DES

Exemple :
P = [5, 3, 1, 4, 2]
A = [0, 1, 1, 0, 0]
P(A) = [0, 1, 0, 0, 1]
Le 5ème bit du tableau A prend la première place, le 3ème bit de A prend la seconde place etc ...

Remarques :
Lorsque l'on fait P(X) = Y, la taille de Y est forcément égale à la taille de P. Il est possible qu'une P-box ne prenne pas tous les éléments de X (P est alors d'une taille inferieure à X) ou alors qu'une P-box prenne des éléments de X en double (P est alors d'une taille plus grande que X).

1.2.3 Permutation circulaire

La permutation circulaire effectue une rotation du tableau de bits vers la droite (permutation circulaire droite) ou vers la gauche (permutation circulaire gauche).
Exemple :
Permutation ciruclaire droite de [1, 2, 3] devient [3, 1, 2].

1.2.4 Substitution (S-Box)

La subsitution permet de passer d'un bloc de n bits à un bloc de p bits. Dans le cas du DES, ce sont des blocs de 6 bits en entrée qui donnent 4 bits en sortie.

S_5		4 bits au centre de l'entrée															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Bits externes	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Exemple d'une S-box

Le bloc de 6-bits est divisé en deux parties, une de 4 bits et une de 2 bits, ce qui donne l'abscisse et l'ordonnée dans le tableau de substitution. Ainsi on a en sortie un bloc de 4-bits.

Exemple :
X = [0,1,1,0,1,1]
S5 : la S-box donnée plus haut
Les bits extrêmes forment alors le couple [0,1] et les bits centraux [1,1,0,1].
Par lecture graphique : S5(X) = [1,0,0,1].
Notons que toutes les tables qui seront utilisées par l'algorithme sont des tables prédéfinies et conçues pour les DES.

2 Fonctionnement du DES

2.1 Principe

Le cryptage d'un bloc est constitué de 4 étapes :
- Diversification de la clef
- Permutation initiale
- Schéma de Feistel
- Permutation finale

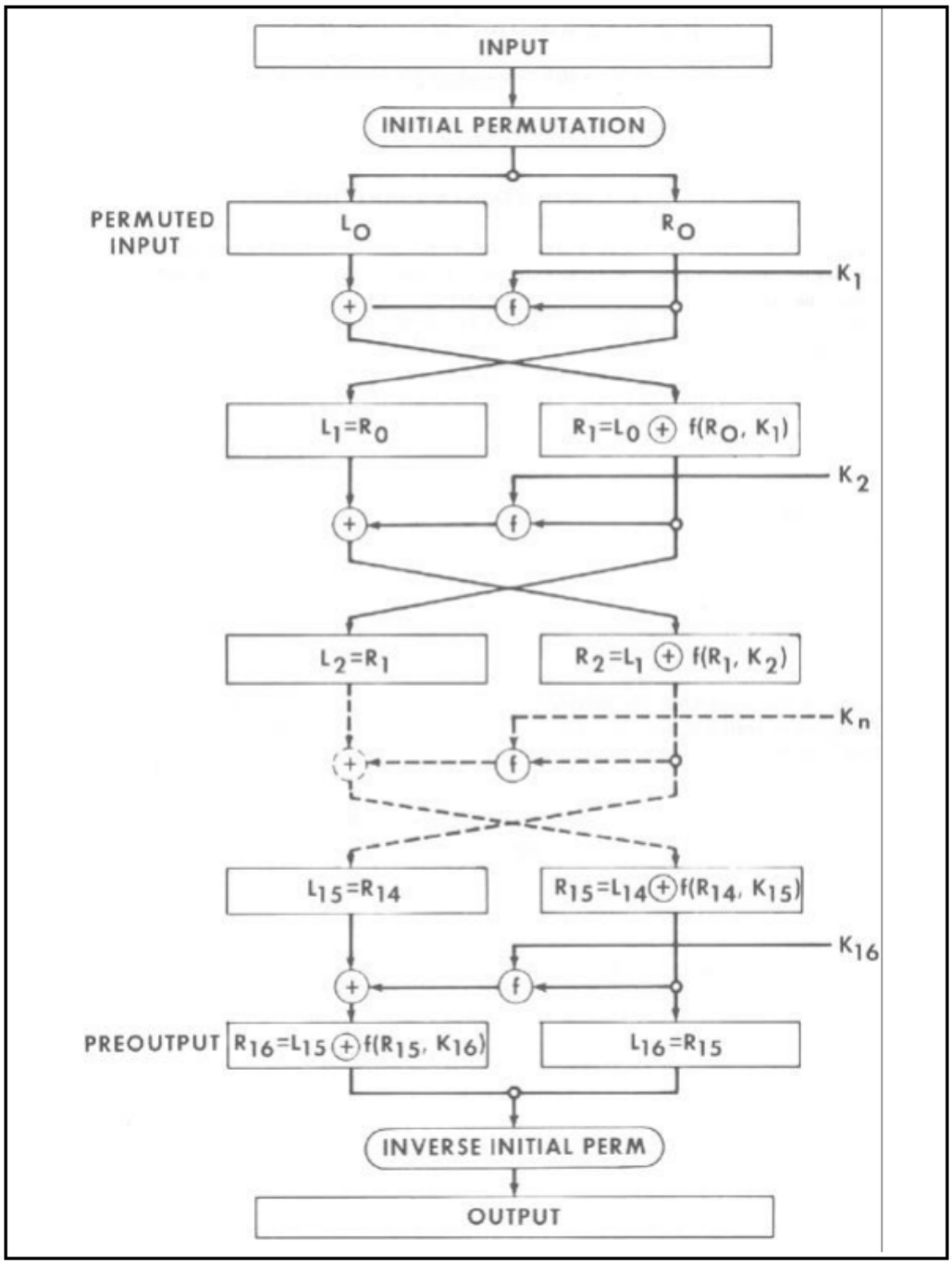

Schéma du DES appliqué sur 1 bloc

2.2 Caractéristiques de la clef

La clef fait 56 bits mais elle est donnée sur 64 bits. Les 8 bits ajoutés sont des bits de parités. Un bit de parité est ajouté tous les 7 bits, de sorte à obtenir une somme impaire de 1.
Exemple :
Avec une liste X de 14 bits (il faut 2 bits de parités), la clef K est :
X = [1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1]
K = [1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1]

2.3 Diversification de la clef

La diversification de la clef utilise la clef de 56 bits, de façon à obtenir 16 sous-clefs de 48 bits qui seront utilisées une par une.

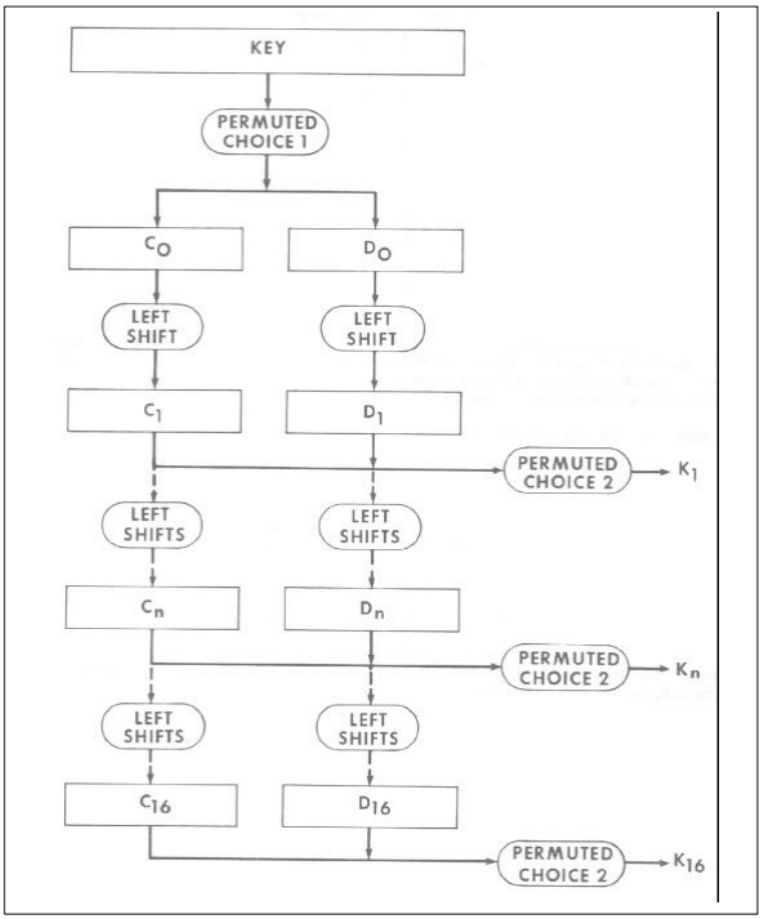

Schéma de la diversification de la clef

Soit K, le bloc de 64 bits représentant la clef initiale.
PC1 une table de permutation de taille 56.
PC2 une table de permutation de taille 48.

Étapes de la diversification de la clef :
- PC1(K) = K' (K' fait 56 bits)
- K' est séparé en C0 et D0 qui sont respectivement la partie gauche (28 premiers bits de K') et droite de K' (28 derniers bits de K')
- $C_1..C_{16}$ et $D_1...D_{16}$ sont déterminés à partir de rotations circulaires de sorte que :
    - G_i = permutation circulaire vers la gauche de G_{i-1}
    - D_i = permutation circulaire vers la gauche de D_{i-1}
    - Si i vaut {1, 2, 9 ,16}, alors on effectue une permutation circulaire supplémentaire sur G_{i-1} et D_{i-1}
- $T_i = C_i + D_i$
- $K_i = PC2(T_i)$

$\{K_1..K_{16}\}$ est l'ensemble des 16 sous-clefs.

2.4 Permutation initiale

Sur le bloc de 64 bits à crypter, est appliquée une table de permutation notée P. Cette table P a sa table inverse notée P^{-1}. Celle-ci permet de remettre les bits dans le bon ordre et de procéder à l'action inverse que fait la table P sur le bloc de bits.

Avec X notre bloc : $P^{-1}(P(X)) = X$

Cette permutation P^{-1} est effectuée sur le bloc à la fin de l'algorithme (la permutation finale).

2.5 Schéma de Feistel

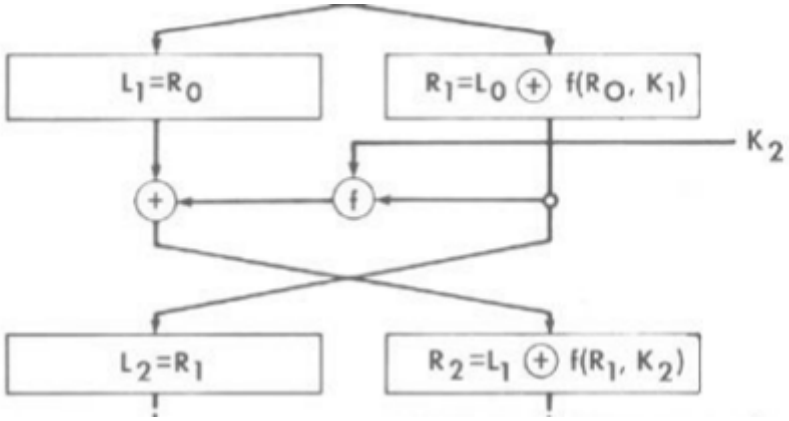

Schéma d'un tour de Feistel

Le schéma de Feistel est une boucle qui consiste à prendre le membre gauche G_i et le membre droit D_i du bloc et de les crypter en utilisant une fonction de confusion f.
A chaque passage de la boucle :
- $L_{i+1} = R_i$
- $R_{i+1} = L_i$ XOR($f(R_i, K_i)$)

Ce tour de boucle se répète pour i allant de 0 à 16 inclus.
La fonction de confusion f consitue le coeur du DES. C'est elle qui assure la sûreté de l'algorithme.

2.6 Permutation finale

La permutation finale permet "d'annuler" la permutation initiale, en remettant dans le bon ordre les bits. Elle est réalisée avec la table P^{-1}. Le bloc final est notre bloc crypté.

2.7 Détails sur la fonction de confusion

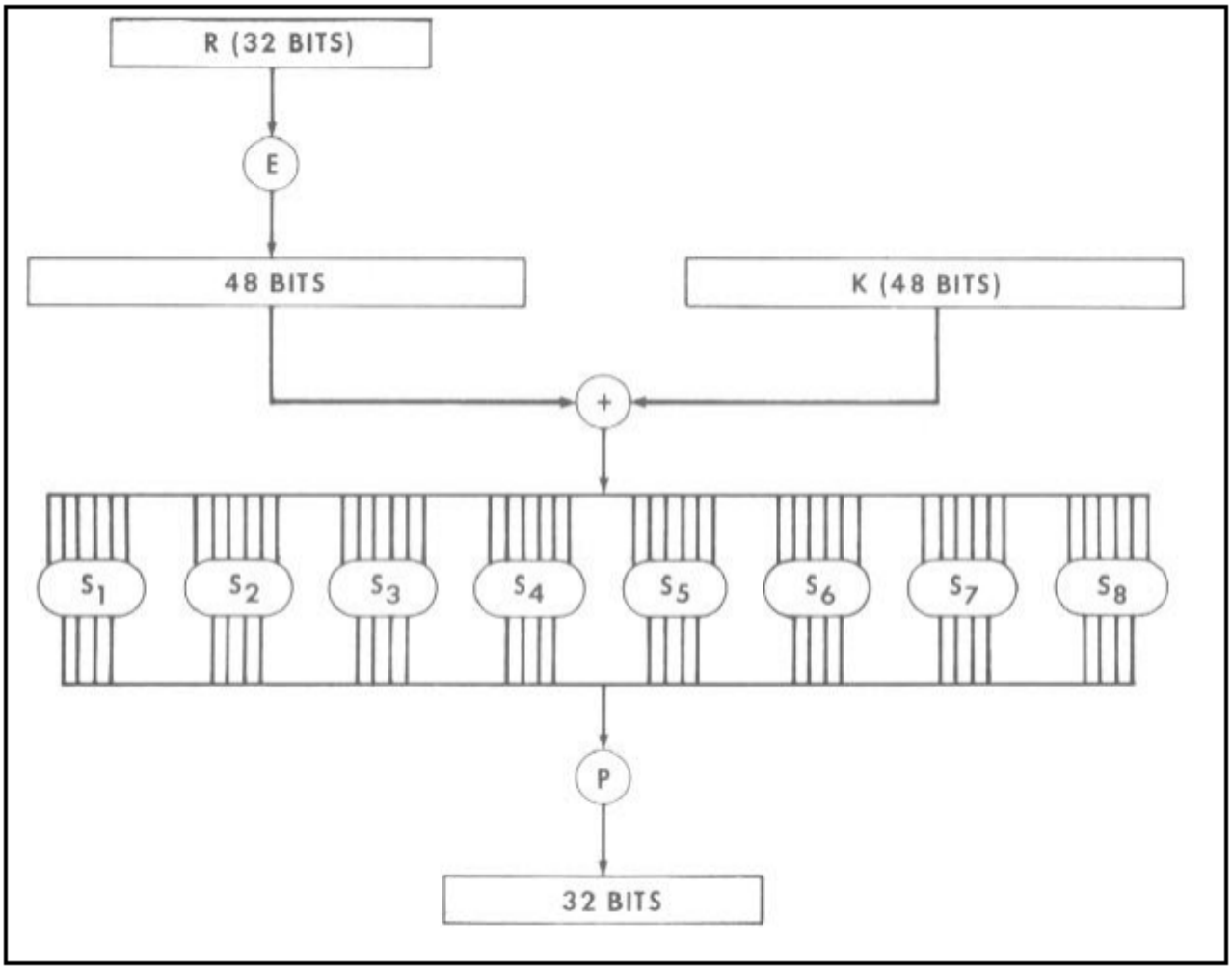

Schéma de la fonction de confusion f

R est le bloc de 32 bits à crypter par f.
K est la clef de 48 bits.
R et K sont donnés en argument à f.
P et E sont des tables de permutation de taille respective 32 et 48 bits.
$\{S_1, S_8\}$ sont 8 S-box. Elles acceptent une entrée de 6 bits et renvoient 4 bits en sortie.

Étapes de la fonction :
- R' = E(R)
- R' XOR K = X
- X est divisé en 8 blocs de 6 bits $[X_1, X_8]$
- Pour chaque X_i : $T_i = S_i(X_i)$
- 8 T_i sont réunis en un bloc Y
- P(Y) = Y'

Le résultat final de f est Y'.

2.8 Remarques sur le décodage

Pour décoder un bloc crypté, il faut :
- Prendre les 16 sous-clefs dans l'ordre inverse
- Inverser les premiers blocs D0 et G0

Ce sont les deux seules différences avec le procédé de cryptage.

3 Code python

3.1 Fonctionnement

Pour chiffrer un fichier, le programme lit le fichier en mode binaire et divise tous les bits en blocs de 4 bits. Si le fichier ne fait pas un nombre exact de blocs de 64 bits, alors le reste de la division euclidienne n'est pas crypté.
Ensuite, tous les blocs sont cryptés les uns après les autres. Enfin, le programme écrase le fichier avec les nouvelles données cryptées. La clef peut être récupérée dans le fichier 'key.txt' créé lors du chiffrement.

3.2 Fonction principale du programme

```
def crypt_bloc(bloc, K, encrypt):
    """Applique l'algorithme DES sur un bloc et encrypte ou décrypte selon les paramètres"""

    c_bloc = permutation(bloc, P)#Permutation initiale
    G,D = str(c_bloc[:32]), str(c_bloc[32:])#G0,D0

    for i in range(16):#Calcul G16,D16 en faisant 16 tours de Feistel
        if encrypt:#Effectue l'algorithme en mode normal (cryptage)
            G, D = Feistel(G, D, K[i])
        else:#Sens inverse de l'algorithme pour le décryptage
            D, G = Feistel(D, G, K[i])


    c_bloc = permutation(G+D, PI)#Permutation finale
    return c_bloc
```

Cette fonction chiffre/déchiffre un bloc de 64 bits. Elle prend comme paramètre le bloc et les 16 sous-clefs ainsi qu'un booléen qui indique si le programme doit chiffrer ou déchiffrer le bloc.
Ensuite, elle fait la permutation initiale, les 16 tours du schéma de Feistel (dans la boucle for) et enfin la permutation finale.

3.3 Différentes fonctions utiles à l'algorithme principal

3.3.1 Permutation

```
def permutation(bits, P):
    """Utilise la P-box pour faire une permutation sur les bits envoyés en paramètre."""
    n_tab = ""
    for i in range(len(P)):
        n_tab += bits[P[i] - 1]#Diminution de 1 car P ne prends pas en compte que les bit
    return n_tab
```

Cette fonction créée un nouveau tableau, dans lequel les bits sont pris selon l'ordre indiqué par la table de permutation P, donnée en paramètre. Notons que l'indice qui est choisi est réduit de 1. En effet, en python les tableaux commencent à l'indice 0, alors que dans les tables de permutation leur premier indice commence à 1.

3.3.2 Permutation circulaire

```
def perm_circ(tab, pas):
    """Décale tout les élément d'un tableau par le pas demandé."""
    n_tab = list(tab)#Copie le tableau (pour avoir un tableau de la même taille)

    for i in range(len(tab)):
        try:
            n_tab[i + pas] = tab[i]
        except IndexError:
            n_tab[(i + pas)%len(n_tab)] = tab[i]#Cas de dépassement de l'index,
    return n_tab
```

Cette fonction créée un nouveau tableau dont les valeurs sont les mêmes que celles du tableau tab donné en paramètre. Toutefois, un décalage des indices est donné par le pas.

3.3.3 Substitution

```
def subs(B, i):
    """Applique la S-box numéro i sur les 6-bits B."""
    colonne = to_decimale(B[1:5])#4 bits centraux
    ligne = to_decimale(B[0] + B[-1])#2 bits extrêmes

    B = to_binary(S[i][ligne][colonne], 4)#4 bits de sortie
    return B
```

Cette fonction calcule en décimal la valeur des 4 bits centraux (colonne de la S-box) et la valeur des deux bits qui sont aux positions extrêmes des 6 bits (ligne de la S-box).
Elle renvoie ensuite la valeur de la case qui est dans la S-box numéro i et dont l'index dépend de la colonne et de la ligne.

3.3.4 XOR

```
def XOR(b1, b2):
    """Renvoi b1 XOR b2.

    b1 et b2 doivent être des strings de bits.

    """
    result = ""

    for i in range(len(b1)):
        if (b1[i] == '1' and b2[i] == '0') or (b1[i] == '0' and b2[i] == '1'):
            result += '1'
        else:
            result += '0'

    return result
```

3.4 Création de la clef aléatoire

```
def create_key():
    """Créé une clef de 64 bits dont les bits 8-16-24...64 sont des bits de parité,
    de sorte que les bytes aient un nombre impaire de 1."""
    key = randbits(56)#Nombre aléatoire de 56 bits
    key = to_binary(key, 56)
    cnt = 0
    sous_key = []
    for i in range(8):
        sous_key.append(key[i * 7 : (i+1) * 7])#Sépare en 8 sous tableaux de 7 bits (8*7 = 56)

    for i in range(len(sous_key)):#Parcourt les sous-tableaux
        for j in range(len(sous_key[i])):
            if sous_key[i][j] == '1':
                cnt += 1#Compteur pour la parité

        sous_key[i] += str( (cnt+1)%2 )#On ajoute un dans le cas où notre nombre est pair, sin
        cnt = 0

    key = ""
    for i in range(len(sous_key)):
        for j in range(len(sous_key[i])):
            key += sous_key[i][j]#Ajoute enfin toutes les sous-clefs en une seule clefs

    return key
```

Randbits est une fonction intégrée à python qui permet de générer un nombre aléatoire d'une certaine longueur de bits. Pour le reste, on ajoute à la fin de chaque bloc de 7 bits, le bit de partité de manière à ce que la somme des bits du bloc (qui fait maintenant 8 bits) soit impaire.
Notons que la fonction randbits fait partie du module "secrets" implémenté dans python 3.6.

4 Triple DES

Le DES n'étant plus un algorithme sûr de nos jours, le triple DES existe et est très simple à mettre en place, une fois qu'on a l'algorithme du DES. Il suffit d'appliquer trois fois l'algorithme DES avec 3 clefs différentes pour chiffrer le fichier.

Ainsi, le nombre de clefs possibles passent de 2^{56} à 2^{168}, ce qui est bien trop colossal pour tenter de déchiffrer le fichier par force brute avec les puissances de calculs actuelles.

Soit E et D, l'action d'encrypter avec le DES. I le bloc à chiffrer/déchiffrer et O le bloc une fois déchiffré/chiffré. Les trois clefs K_1, K_2 et K_3.

Le fonctionnement du triple DES est :

Chiffrement : $O = E_{K3}(D_{K2}(E_{K1}(I)))$.

Déchiffrement : $O = D_{K1}(E_{K2}(D_{K3}(I)))$

Le triple DES est une combinaison de E et de D. Lors du déchiffrage du bloc, il suffit de faire les actions contraires et dans le sens inverse.

Cet algorithme a vu le jour en 1998, lorsque le DES n'était plus un algorithme de chiffrement sûr. Il a l'avantage d'augmenter la taille de la clef sans avoir à créer un tout nouvel algorithme. En revanche, il présente l'inconvénient de nécéssité un temps d'exécution trois fois plus long.
Il est encore actuellement utilisé par certains logiciels de Microsoft (source Wikipédia).

Sources

- http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/des : site regroupant plusieurs techniques de cryptographie et consituant une bonne base pour le DES.
- http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf : publication officielle sur le DES contenant toutes les tables.