Corrigé : chiffrement par blocs (d'après X mp 2009)

Partie I. Chiffrement et déchiffrement

Question 1. $a_0 \in [\![0,1[\![$ donc $a_0 = 0$.
$a_1 \in [\![0,2[\![$ et $k - a_1$ est pair donc $a_1 = k \bmod 2$.
$a_2 \in [\![0,3[\![$ et $\frac{1}{2!}(k-a_1) - a_2$ est divisible par 3 donc $a_2 = \frac{1}{2!}(k-a_1) \bmod 3$.
Plus généralement, $\frac{1}{i!}\Big(k - \sum_{j=0}^{i-1} a_j j!\Big) \equiv a_i \bmod (i+1)$, donc $a_i = \frac{1}{i!}\Big(k - \sum_{j=0}^{i-1} a_j j!\Big) \bmod (i+1)$.
Ceci conduit à définir :

```
def decomposerFact(N, k):
    a = [0] * N
    for i in range(1, N):
        a[i] = k % (i + 1)
        k //= (i + 1)
        if k == 0:
            break
    return a
```

Question 2.

```
def ecrirePermutation(N, k):
    a = decomposerFact(N, k)
    ell = [i for i in range(N)]
    sigma = [None] * N
    for i in range(N):
        sigma[i] = ell[a[N-i-1]]
        del ell[a[N-i-1]]
    return sigma
```

La fonction `del` supprime un élément d'un tableau donné par son rang.

Question 3. La fonction de chiffrement est immédiate :

```
def chiffrer(N, k, b):
    sigma = ecrirePermutation(N, k)
    return sigma[b]
```

La fonction de déchiffrement l'est tout autant si on connait la méthode `index(x)` qui, appliquée à une liste, renvoie l'indice correspondant à la première occurrence de `x` dans celle-ci :

```
def dechiffrer(N, k, b):
    sigma = ecrirePermutation(N, k)
    return sigma.index(b)
```

À défaut on peut écrire :

```
def dechiffrer(N, k, b):
    sigma = ecrirePermutation(N, k)
    for i in range(N):
        if sigma[i] == b:
            return i
```

Partie II. Réseau de Feistel

On notera que le choix de $N = 2^{64}$ n'est pas anodin : les entiers étant codés en complément à deux sur 64 bits, le quotient de la division euclidienne par 2^{32} est égal au 32 premiers bits de cette décomposition et le reste aux 32 derniers.

Question 4.

```
def oplus(x, y):
    z = 0
    p = 1
    while x > 0 or y > 0:
        if x % 2 != y % 2:
            z += p
        x, y = x // 2, y // 2
        p *= 2
    return z
```

Notons que cette fonction est prédéfinie en Python : il s'agit de l'opérateur infixe `^`.

Question 5.

```
def feistelTour(k, b):
    q, r = divmod(b, 2**32)
    return r * 2**32 + oplus(q, f(k, r))
```

`q, r = divmod(a, b)` est équivalent à `q, r = a // b, a % b` (mais ne réalise qu'une seule fois le calcul de la division euclidienne).

Question 6. La relation $(a \oplus b) \oplus b = a$ permet d'établir les équivalences suivantes :

$$\begin{cases} r_{i+1} = q_i \oplus F_{k_i}(r_i) \\ q_{i+1} = r_i \end{cases} \iff \begin{cases} q_i = r_{i+1} \oplus F_{k_i}(r_i) \\ r_i = q_{i+1} \end{cases} \iff \begin{cases} q_i = r_{i+1} \oplus F_{k_1}(q_{i+1}) \\ r_i = q_{i+1} \end{cases}$$

```
def feistelInverseTour(k, b):
    q, r = divmod(b, 2**32)
    return oplus(r, f(k, q)) * 2**32 + q
```

Question 7.

```
def feistel(K, b):
    for k in K:
        b = feistelTour(k, b)
    return b
```

Question 8.

```
def feistelInverse(K, b):
    for k in reversed(K):
        b = feistelInverseTour(k, b)
    return b
```